Security Administration and Programming Guide

EAServer

5.0

DOCUMENT ID: DC38035-01-0500-01

LAST REVISED: December 2003

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

About This Book

This book describes the features in EAServer with which you can define the security characteristics of client/server communications.

Audience

Use this document if you are responsible for creating or deploying secure components, applications, and Web applications, or for defining secure EAServer listeners with which clients communicate.

How to use this book

Use this document to understand EAServer security.

The contents of this book are:

- Chapter 1, “Security Concepts” – provides an overview of security terms and concepts and describes how to meet the challenge of protecting server resources.
- Chapter 2, “Securing Component Access” – describes how to authenticate base clients, other components, or servlets and JSPs. Also describes how to pass credentials from EJBs and servlets between servers.
- Chapter 3, “Using Web Application Security” – Describes how to secure Web applications and the resources contained within Web applications.
- Chapter 4, “Securing TDS Client Access” – describes how TDS and MASP clients can securely communicate with EAServer.
- Chapter 5, “Using SSL in Java Clients” – describes how to use SSL in Java clients.
- Chapter 6, “Using SSL in C++ Clients” – describes how to use SSL in C++ clients.
- Chapter 7, “Using SSL in PowerBuilder Clients” – describes how to use SSL in PowerBuilder® clients.
- Chapter 8, “Using SSL in ActiveX Clients” – describes how to use SSL in ActiveX clients.

- Chapter 9, “Creating and Using Custom Security Components” – describes how to create and implement custom role and service components to meet your specific authentication and authorization needs.
- Chapter 10, “Using the JAAS API” – describes how to implement the Java Authentication and Authorization Support (JAAS) module in clients, EAServer, and as connectors to other servers.
- Chapter 11, “Deploying Applications Around Proxies and Firewalls” – describes how to deploy applications around firewalls and how to use reverse proxies.
- Chapter 12, “Security Configuration Tasks” – describes the major security tasks you perform from EAServer Manager, including:
  - Role mapping
  - OS-based authentication
  - Defining security profiles that use SSL
  - Assigning security profiles to EAServer listeners
- Chapter 13, “Managing Keys and Certificates” – describes how to manage all aspects of SSL keys and certificates.
- Chapter 14, “Entrust PKI Integration” – describes how to use the Entrust public-key infrastructure (PKI) for secure client/server communication.
- Chapter 15, “Tutorial: Using SSL” – steps you through the process of using SSL in a browser and EAServer for secure communication.

Conventions

The formatting conventions used in this manual are:

Formatting example	To indicate
commands and methods	When used in descriptive text, this font indicates keywords such as: • Command names used in descriptive text • C++ and Java method or class names used in descriptive text • Java package names used in descriptive text • Property names in the raw format, as when using jagtool to configure applications rather than EAServer Manager

Formatting example	To indicate
variable, *package*, or *component*	Italic font indicates: • Program variables, such as *myCounter* • Parts of input text that must be substituted, for example: *Server*.log • File names • Names of components, EAServer packages, and other entities that are registered in the EAServer naming service
File | Save	Menu names and menu items are displayed in plain text. The vertical bar shows you how to navigate menu selections. For example, File | Save indicates "select Save from the File menu."
`package 1`	Monospace font indicates: • Information that you enter in EAServer Manager, a command line, or as program text • Example program fragments • Example output fragments

Related documents

Core EAServer documentation The core EAServer documents are available in HTML format in your EAServer software installation, and in PDF and DynaText format on the *Technical Library* CD.

What's New in EAServer summarizes new functionality in this version.

The *EAServer Cookbook* contains tutorials and explains how to use the sample applications included with your EAServer software.

The *EAServer Feature Guide* explains application server concepts and architecture, such as supported component models, network protocols, server-managed transactions, and Web applications.

The *EAServer System Administration Guide* explains how to:

- Start the preconfigured Jaguar server and manage it with the EAServer Manager plug-in for Sybase Central™
- Create, configure, and start new application servers
- Define connection caches
- Create clusters of application servers to host load-balanced and highly available components and Web applications
- Monitor servers and application components
- Automate administration and monitoring tasks with jagtool

The *EAServer Programmer's Guide* explains how to:

- Create, deploy, and configure components and component-based applications
- Create, deploy, and configure Web applications, Java servlets, and JavaServer Pages
- Use the industry-standard CORBA and Java APIs supported by EAServer

The *EAServer Web Services Toolkit User's Guide* describes Web services support in EAServer, including:

- Support for standard Web services protocols such as Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Uniform Description, Discovery, and Integration (UDDI)
- Administration tools for deployment and creation of new Web services, WSDL document creation, UDDI registration, and SOAP management

The *EAServer Performance and Tuning Guide* describes how to tune your server and application settings for best performance.

The *EAServer API Reference Manual* contains reference pages for proprietary EAServer Java classes, ActiveX interfaces, and C routines.

The *EAServer Troubleshooting Guide* describes procedures for troubleshooting problems that EAServer users may encounter. This document is available only online; see the EAServer Troubleshooting Guide at http://www.sybase.com/detail?id=1024509.

Message Bridge for Java™ Message Bridge for Java simplifies the parsing and formatting of structured documents in Java applications. Message Bridge allows you to define structures in XML or other formats, and generates Java classes to parse and build documents and messages that follow the format. The *Message Bridge for Java User's Guide* describes how to use the Message Bridge tools and runtime APIs. This document is included in PDF and DynaText format on your *EAServer 5.0 Technical Library* CD.

Adaptive Server Anywhere documents EAServer includes a limited-license version of Adaptive Server Anywhere for use in running the samples and tutorials included with EAServer. Adaptive Server Anywhere documents are available on the Sybase Web site at http://sybooks.sybase.com/aw.html.

jConnect for JDBC documents EAServer includes the jConnect™ for JDBC™ driver to allow JDBC access to Sybase database servers and gateways. The *Programmer's Reference jConnect for JDBC* is available on the Sybase Web site at http://sybooks.sybase.com/jc.html.

Accessibility features

EAServer 5.0 has been tested for compliance with U.S. government Section 508 Accessibility requirements. The online help for this product is also provided in HTML, JavaHelp, and Eclipse help formats, which you can navigate using a screen reader.

EAServer Manager supports working without a mouse. For more information, see "Keyboard navigation" in Chapter 2, "Sybase Central Overview," in the *EAServer System Administration Guide*.

The WST plug-in for Eclipse supports accessibility features for those that cannot use a mouse, are visually impaired or have other special needs. For information about these features refer to Eclipse help:

1 Start Eclipse

2 Select Help | Help Contents

3 Enter Accessibility in the Search dialog box

4 Select Accessible user interfaces or Accessibility features for Eclipse

---

Note You might need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and MixedCase Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool.

For additional information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

Other sources of information

Use the Sybase Getting Started CD, the Sybase Technical Library CD and the Technical Library Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the Technical Library CD. It is included with your software. To read or print documents on the Getting Started CD you need Adobe Acrobat Reader (downloadable at no charge from the Adobe Web site, using a link provided on the CD).
- The Technical Library CD contains product manuals and is included with your software. The DynaText reader (included on the Technical Library CD) allows you to access technical information about your product in an easy-to-use format.

Refer to the *Technical Library Installation Guide* in your documentation package for instructions on installing and starting the Technical Library.

- The Technical Library Product Manuals Web site is an HTML version of the Technical Library CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Updates, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Technical Library Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals/.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ **For the latest information on product certifications**

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Select Products from the navigation bar on the left.
3. Select a product name from the product list.
4. Select the Certification Report filter, specify a time frame, and click Go.
5. Click a Certification Report title to display the report.

❖ **For the latest information on EBFs and Updates**

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Select EBFs/Updates. Enter user name and password information, if prompted (for existing Web accounts) or create a new account (a free service).
3. Specify a time frame and click Go.
4. Select a product.
5. Click an EBF/Update title to display the report.

❖ **To create a personalized view of the Sybase Web site (including support pages)**

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click MySybase and create a MySybase profile.

Accessibility features

EAServer 5.0 has been tested for compliance with U.S. government Section 508 Accessibility requirements. The online help for this product is also provided in HTML, JavaHelp, and Eclipse help formats, which you can navigate using a screen reader.

EAServer Manager supports working without a mouse. For more information, see "Keyboard navigation" in Chapter 2, "Sybase Central Overview," in the *EAServer System Administration Guide*.

The WST plug-in for Eclipse supports accessibility features for those that cannot use a mouse, are visually impaired or have other special needs. For information about these features refer to Eclipse help:

1 Start Eclipse

2 Select Help | Help Contents

3 Enter Accessibility in the Search dialog box

4 Select Accessible user interfaces or Accessibility features for Eclipse

---

Note You might need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and MixedCase Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool.

For additional information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1

Security Concepts

Keeping resources secure is an ongoing challenge. As defenses are implemented, new methods are devised to circumvent them. The following is a short list of the many excellent sites that contain security related information:

- The JavaWorld's bookstore security reference page at http://www.javaworld.com/javaworld/books/jw-books-security.html
- A good source of cryptographic related information is available from RSA laboratories at ftp://ftp.rsasecurity.com/pub/labsfaq/labsfaq4.pdf.
- The Java security page at http://java.sun.com/security/

Authentication and authorization

Authentication means that an entity's (person, client, or server) identity has been verified to either a server or a client. In contrast, authorization means that an entity has permission to use a resource or file. An entity must be authenticated before it can be authorized to use a resource or file. This book describes authentication and authorization services provided for:

- Components and packages
- Web clients
- Java, C++, and ActiveX clients
- TDS and MASP clients

- Web applications
- Client/application server connections

Public-key cryptography

To maintain secure communications between a client and host, public-key cryptography techniques are used for:

- Authentication – verifying the identity of both the client and the server; Public-key cryptography techniques use digitally signed certificates that identify network entities.
- Encryption – modifying data so that it can be read only by the party for whom it is intended. When used with a user's private key, certificates encrypt and decrypt messages.

Unencrypted messages are known as *plain text*. Encoding the contents of a message is called *encryption*. This encrypted message is the *cipher text*. *Decryption* is the process of retrieving the plain text from the cipher text. A key is usually required to perform encryption and decryption. A *CipherSuite* defines the parameters and methods supported by both the client and server that perform the encryption and decryption.

Public-key encryption uses a pair of keys for encryption and decryption. One key is secret (the private key) and the other is distributed (the public key). You send your digitally signed public key (certificate) to anyone with whom you wish to communicate using encoded data.

Messages that are sent to you are encrypted with your distributed public key and decrypted by your private key, while messages sent by you are encrypted with your private key and decrypted with your distributed public key. *RSA encryption* is a widely used public-key encryption system.

For more information on RSA and public-key encryption, see the RSA Web site at http://www.rsa.com .

Public-key certificates

Public-key certificates provide a method to identify and authenticate clients and servers on the Internet. Public-key certificates are administered and issued by a third party known as a *certification authority* (CA). A *subject* (individual, system, or other entity on the network) uses a program to generate a key pair and submits the public key to the CA along with identifying information (such as name, organization, e-mail address, and so on). This is known as a *certificate request*. The CA issues a digitally signed certificate. A *digital signature* is a block of data that is created using a private key.

The CA ties the certificate owner to the public key within the certificate. The subject then uses the certificate, along with his private key to establish his identity. Once this is done, whomever the subject is communicating with knows that a third party has vouched for his identity.

The process requires these steps:

1 A client submits a request for, and receives, a certificate from the CA.

2 An administrator installs the CA's certificate on the server and marks it trusted. Any client certificate signed by the same CA will now be trusted and accepted by the server.

3 The client supplies its certificate and negotiates a secure connection with the server.

SSL, HTTPS, and IIOPS

SSL provides security for network connections. Specifically, SSL uses public-key encryption to provide:

- Client and server authentication using certificates
- Encryption, which prevents third parties from understanding transmitted data
- Integrity checking, which detects whether transmitted data has been altered

Packets for other protocols can be embedded inside of SSL packets. A connection in which the application protocol is embedded inside of SSL is an *SSL-tunnelled* connection.

Both IIOP and HTTP can be tunnelled inside SSL, which means that these protocols take advantage of SSL security features. For example, HTTPS connections embed HTTP packets inside of SSL packets. Your Web browser creates a secure HTTP connection any time you load a page from a URL that begins with “https:”

See the following SSL-related chapters:

- Chapter 5, “Using SSL in Java Clients”
- Chapter 6, “Using SSL in C++ Clients”
- Chapter 8, “Using SSL in ActiveX Clients”
- Chapter 13, “Managing Keys and Certificates”
- Chapter 15, “Tutorial: Using SSL”

Proxies and firewalls

A *firewall* is a system that enforces an access control policy between networks. Located on a gateway into the network, the firewall blocks traffic that does not have permission to access the network. An organization establishes a firewall so that it can control access to resources. For example, an organization that allows intranet users access to the Internet installs a firewall to prevent external users from accessing internal resources.

Proxy servers are typically used to constrain and secure connections from an organization’s computers to sites that require connecting across the Internet. To enhance security, some network configurations require all Internet connections to go through a proxy server, including IIOP connections to an application server.

See Chapter 11, “Deploying Applications Around Proxies and Firewalls” for more information.

Lines of defense

This section describes types of attacks and some strategies for defending against them.

Types of attacks

There are several ways in which data can be tampered with, compromised, and stolen. In addition, systems can be overwhelmed with traffic to the point that they are rendered useless.

Integrity attacks Data integrity is a measure of the quality of the information stored and transmitted on a system.

Types of attacks on data integrity include deleting or modifying files or information on the file system or over a network.

Spoofing IP spoofing occurs when an intruder attempts to deceive the target system into accepting packets that appear to the target as coming from someone other than the intruder. If the target system already has an authenticated TCP session with another system and mistakenly accepts spoofed IP packets, the intruder can access sensitive information and lead the target to execute commands in that packet, as though they came from the authenticated connection.

Availability attacks Availability attacks occur when a resource such as a Web site or HTTP port becomes unavailable due to a high volume of traffic. Someone can use a program to generate thousands of simultaneous requests aimed at the same site which then is unable to respond to legitimate requests.

Capture-and-replay Capture-and-replay refers to an intruder capturing data as it moves from one system to another. User names, passwords, authentication information, and so on, can be tampered with or used by the intruder to gain access to protected resources.

There are a variety of ways and tools that intruders use to gain access to system resources. Some of these attacks are undetected, while others destroy or alter information. Following is a few examples of how an intruder gains access to system resources:

- A *brute force* attack involves using many combinations until the right key/password is located. Although it may seem like an expensive operation, both in time and resources, tools are available that can speed-up the process.
- A *trojan horse* attack occurs when an intruder secretly inserts a program or file that either steals or destroys information, such as a virus. Another simple example would be for someone to place a bogus program on your system that prompts for a user name and password. The program simply logs the user name and password information. The intruder accesses this information and can then use your user name and password to access resources to which you are permitted.

- A *person-in-the-middle* attack intercepts communication between two parties without their knowledge. This attack allows two parties to communicate without knowing a third party has access to the same information.

Defense against attacks

This section discusses some of the methods by which you can protect data and restrict access to resources.

Protecting ports and listeners You can provide various levels of security to EAServer listeners by assigning security profiles to HTTPS and IIOPS listeners. See Chapter 12, “Security Configuration Tasks” for more information.

Protecting application server resources and securing clients EAServer provides several methods to protect server resources and secure client/server connections:

- Set authentication and authorization levels using EAServer Manager. See Chapter 3, “Using Web Application Security.”
- Create custom authentication and authorization components. See Chapter 9, “Creating and Using Custom Security Components.”
- Use the Java authentication and authorization service (JAAS). See Chapter 10, “Using the JAAS API.”
- Use SSL to protect your Java, C++, and ActiveX clients. See Chapter 5, “Using SSL in Java Clients,” Chapter 6, “Using SSL in C++ Clients,” and Chapter 8, “Using SSL in ActiveX Clients”.
- Establish minimum levels of protection for components, packages, and methods using quality of protection (QOP). See Chapter 2, “Securing Component Access.”
- Propagate client principal information from one server to another and use run-as support so an EJB can perform method invocations on other EJBs using a different identity. See Chapter 2, “Securing Component Access.”

Protecting data Use public-key certificates when exchanging sensitive data over a network to protect it from being viewed by intruders. See Chapter 13, “Managing Keys and Certificates” for more information.

CHAPTER 2

Securing Component Access

Components can be invoked by clients, other components, or servlets and JSPs. This chapter describes the various methods used to authenticate and authorize each type of client.

Client authentication

Users are authenticated when a client application creates a proxy or stub object (a connection is made when the application creates the first proxy or stub; other proxies or stubs may use the same connections or allocate new connections as needed).

Authentication options for base clients include:

- **No authentication** No user name/password authentication of session user names is performed. This is the default configuration for new servers.
- **Native operating system authentication** User names for an EAServer connection map directly to a login name on the host operating system. For example, for UNIX, you would use network information service (NIS) passwords, and for Windows, you would use your Windows domain password. See “Configuring OS authentication” on page 137 for more information. You can enable native authentication with EAServer Manager using the server’s property sheet.

- **SSL certificate-based authentication** You can configure a secure IIOP port that requires mutual (client and server) authentication. Clients must have a valid SSL certificate to connect to this port, and the certificate must be issued by a certificate authority that is trusted by EAServer.

  When clients connect with an SSL client-side certificate, the client also supplies an EAServer user name and password for the connection in addition to the certificate. EAServer performs authorization checking based on the EAServer user name. The SSL user name and certificate information are available through the built-in CtsSecurity/SessionInfo component.

  EAServer provides native SSL support without the use of proxies. On the client side, the EAServer Java ORB supports SSL when run in Netscape 4.0. Java applets and Java applications, C++, PowerBuilder, and ActiveX components can use SSL natively. Other types of clients require the use of an SSL proxy.

  C++ and PowerBuilder clients require that a public-key infrastructure (PKI) system be available on the client to manage digital certificates. You can use EAServer Manager, which manages EAServer's certificate database, or you can use Entrust/Entelligence, available separately from Entrust Technologies at http://www.entrust.com.

  See "Configuring security profiles" on page 139 for information about the various authentication levels you can establish for a client-EAServer connection. See Chapter 14, "Entrust PKI Integration" for Entrust information.

- **Custom authentication** You can code a service component to be installed in EAServer to perform your own authentication checks. For example, you can retrieve the client user name and password and check to see if they allow a login to a remote database server. See "User installed authentication services" on page 9 for more information.

- **Quality of protection** EAServer Manager allows you to set the quality of protection (QOP) for EAServer packages, components, and methods. QOP establishes a minimum level of encryption and authentication that a client must meet before it can access your business logic. See "Quality of protection" on page 18 for more information.

EAServer provides a special user name, jagadmin, for the Jaguar Administrator login. Administrator authentication is performed independently of the authentication option you configure. By default, the "jagadmin" user name has no password.

Set the administrator password for new servers Immediately after you create a new server, you must secure access to the server by defining the jagadmin password and configuring the authentication mechanism of your choice. See "Administration password and OS authentication" in the *EAServer System Administration Guide* for more information.

User installed authentication services You can install your own service component to authenticate clients for any EAServer. For example, if you require the client user name to match a remote database user name, you can code the component to retrieve the client user name and password and attempt to log in to the remote database. For more information, see Chapter 9, "Creating and Using Custom Security Components."

Intercomponent authentication

This section describes various security features available to components, including:

- Retrieving SSL information
- Restricting access to EJBs
- Authenticating non-EJB components within a server and for standalone clients
- Issuing intercomponent calls using SSL

Accessing SSL information

Clients can connect to a secure IIOP port using an SSL client certificate. You can issue intercomponent calls to the built-in *CtsSecurity/SessionInfo* component to retrieve the client certificate data, including:

- The distinguished SSL user name
- The client certificate fingerprint (MD5 message digest)
- The client certificate data
- The chain of issuing certificates

This component implements CtsSecurity::SessionInfo IDL interface. HTML documentation is available for the interface in the *html/ir* subdirectory of your EAServer installation. You can view it by loading the main EAServer HTML page, then clicking the “Interface Repository” link.

The CtsSecurity::UserCredentials interface is deprecated The CtsSecurity::UserCredentials interface, which is implemented by the *CtsSecurity/UserCredentials* component, has been replaced by the CtsSecurity::SessionInfo interface, which provides additional functionality such as certificate parsing. EAServer supports the CtsSecurity::UserCredentials interface for backward compatibility. Use the CtsSecurity::SessionInfo interface if you are developing new components.

Non-EJB components

For non-EJB CORBA components, the following mechanisms are used for authentication within a server and for standalone clients:

1 Embed the user name and password in the URL when creating a component instance. For example:

```
Module::Interface_var compInstance = Module::Interface::narrow(
"iiop[s]://user:password:host:port/EAServerPackage/EAServerComponent");
```

2 Use the lookup method on SessionManager::Factory. You cannot embed a user name/password in the URL.

See the SessionManager IDL documentation for more information and these chapters:

- Chapter 5, “Using SSL in Java Clients”
- Chapter 6, “Using SSL in C++ Clients”
- Chapter 8, “Using SSL in ActiveX Clients”

C++ components

C++ components (and PowerBuilder NVOs) can make intercomponent calls across different servers using SSL in much the same way as any other C++ client. However, be aware of these considerations:

- The SSLServiceProvider interface is not available to components. Instead, set ORB-level SSL properties to initiate server-to-server intercomponent calls using SSL.
- Components use *$JAGUAR/Repository/Security* path to locate certificates and key database files if not using Entrust IDs. That is, components making intercomponent calls use the EAServer's certificate and key database managed by EAServer Manager.

For information about developing C++ components and clients, see these chapters in the *EAServer Programmer's Guide*:

- Chapter 14, "Creating CORBA C++ Components"
- Chapter 15, "Creating CORBA C++ Clients"

Your EAServer installation includes a sample C++ component that demonstrates how to call the *CtsSecurity/SessionInfo* component methods. See the following file in your EAServer installation for more information:

```
sample/SecurityDemo/readme.txt
```

Intercomponent authentication for EJBs and servlets

EAServer 4.0 implements J2EE version 1.3 security requirements, including Java and C++ ORB support and CORBA Secure Interoperable version 2 protocol (CSIv2). CSIv2 is part of EJB version 2.0 interoperability requirements, and supports:

- EJB 2.0 security features including:
  - Caller propagation on remote servers from EJB 2.0 clients using RMI/IIOP.
  - Run-as support
  - Trust identities
- Servlet 2.3 security enhancements including:
  - Caller propagation on remote servers
  - Run-as support
- Java Authentication and Authorization Service (JAAS) – see Chapter 10, "Using the JAAS API" for more information.

Other references

For more information about J2EE version 1.3, see the Java Web site at http://java.sun.com/j2ee.

For more information about servlet technology, see the Java Web site at http://java.sun.com/products/servlet/index.html.

For more information about CSIv2, see the OMG Web site at http://www.omg.org/technology/documents/formal/omg_security.htm

Intercomponent authentication for EJB 2.0 components

EJB 2.0 components use caller propagation to pass client information between servers for authentication, whereas run-as support allows EJB 2.0 components to perform method invocations on other components using a different identity.

Caller propagation

Caller propagation allows an EJB 2.0 RMI/IIOP client to pass principal information to a server and have that information propagated to other servers. In other words, EAServer 4.0 can pass a client's user name or X.509 certificate information from an EJB on one server, to an EJB on a different server. For example:

1 The client passes principal information to EAServer1, where the information is authenticated.

2 EAServer1 retrieves the remote client's authentication information by calling getCallerPrincipal().

3 EJBA, on EAServer1, makes a call to another Bean, which resides on EAServer2.

4 The propagated caller information is retrieved on EAServer2 using the getCallerPrincipal() method.

To enable caller propagation for EJB component calls made in servlet or component code, you must specify a `corbaname` URL in the EJB Reference properties for the EJB component, servlet, or JSP that issues the call.

For information on interoperable naming URLs, see Chapter 9, "EAServer EJB Interoperability," in the *EAServer Programmer's Guide*.

Run-as support

Normally, when a component calls another component, the invocation uses the client's credentials. You can use *identities* to specify alternate credentials for intercomponent calls. Identities map logical identity names to a user name, password, and required SSL session characteristics. The identity names are used in the run-as mode settings for components and component methods.

Run-as support enables an EJB 2.0 component to perform method invocations on other components using a specified identity. This identity can be configured at deployment time. In the standard EJB 2.0 deployment descriptor, the run-as property is expressed in terms of a role. The role is a name of a security-role element defined in the same deployment descriptor. It is expected that at deployment time, or when configuring a new EJB, the role name should be defined. Further, the deployer selects a Jaguar identity that is expected to be present in this role. This Jaguar identity is used while invoking another EJB. The run-as feature can be enabled via EAServer Manager.

To enable use of the run-as identity for EJB component calls made in component code, you must specify `corbaname` URLs in the EJB Reference properties for the EJB component that issues the call. For information on interoperable naming URLs, see Chapter 9, "EAServer EJB Interoperability," in the *EAServer Programmer's Guide*.

❖ **Configuring an EJB 2.0 component to run as a different identity**

1 If necessary, define the identity to be used as described in "Configuring identities" on page 149.

2 Highlight the EJB 2.0 component for which you are establishing a run-as identity.

3 Display the Run As Identity tab and configure the settings as follows:

- Run as – choose `specified`.
- Role – specify a role name. The identity specified in the Mapped to Jaguar identity field should be in this role. This name is used if the component is exported to an EJB-JAR file.
- Run as identity – specify a logical identity name.
- Mapped to Jaguar identity – choose an EAServer identity from the pull down menu. This is the identity with which the component executes.

- Description – enter an optional text comment. This field can be used to provide identity mapping instructions for the deployer when the component is deployed to another server.

The Existing Mappings on the Package table displays logical identity names that are mapped to EAServer identities by components in the same package.

You can configure a run-as identity application or server-wide. This provides a convenient way to globally set the run-as identity for all of the EJBs in an application or server.

❖ **Configuring EJB 2.0 components or servlets to run as a different identity at the application or server level**

1 If necessary, define the identity to be used as described in "Configuring identities" on page 149.

2 Select the server or application for which you are configuring the run-as identity.

3 Select File | Server Properties or File | Application Properties.

4 Select the Security tab.

5 For a server, click the Set Trusted and Security Identities button. Select the run-as identity from the Run-as Identity drop-down list.

To set the run-as identity application-wide, select the run-as identity from the Run-as Identity drop-down list.

You can check the setting of your run-as identity from the Advanced tab by viewing the `com.sybase.jaguar.server.security.runasidentity` property, and the `com.sybase.jaguar.application.security.runasidentity` property. Do not set the run-as identity in the Advanced tab since these values are overwritten by the values set in the Security tab.

Trusted identities

Identities defined in EAServer Manager configure client identities (user names, SSL certificates, or Entrust users) that can be assumed by executing components. For caller propagation, EAServer requires an identity to propagate a remote client's credentials to another server when it cannot include, as part of the request, the client's authentication data (password or a private key corresponding to a X.509 certificate). You configure a server (or container) to trust a set of identities that vouch for the client. These identities are known as trusted identities.

If a target server trusts an intermediate server, it is implied that the target server trusts all servers trusted by the intermediate server.

A server or container needs to establish a list of identities it trusts. Servers and containers use identities for the purpose of authentication. Other servers need to know the list of trusted identities for a server while connecting to it.

Configuring an identity for outgoing credential propagation

An identity is required when a server is making remote IIOP or IIOPS connections to other servers, and is not necessary for in-server or in-memory component calls. Use EAServer Manager to establish this identity at the server or application level.

❖ **Configuring a security identity for outgoing interserver calls**

1 If necessary, define the identity to be used as described in "Configuring identities" on page 149.

2 Select the server or application for which you are configuring the security identity.

3 Select File | Server Properties or File | Application Properties.

4 Select the Security tab.

5 For a server, click the Set Trusted and Security Identities button. Select the security identity from the Security Identity drop-down list.

   For an application, select the security identity from the Security Identity drop-down list.

You can check the setting of your security identity from the Advanced tab by viewing the `com.sybase.jaguar.server.security.identity` property, and the `com.sybase.jaguar.application.security.identity` property. Do not set the security identity in the Advanced tab since these values are overwritten by the values set in the Security tab.

Enabling trusted identities on the peer

A trusted identity vouches for someone else and is always authenticated by the peer. Establish a list of trusted identities at the server or application level.

❖ **Establishing a list of trusted identities for incoming interserver calls**

1. If necessary, define identities to be trusted as described in "Configuring identities" on page 149.
2. Select the server or application for which you are establishing trusted identities.
3. Select File | Server Properties or File | Application Properties.
4. Select the Security tab.
5. For a server, click the Set Trusted and Security Identities button. Click the Add button and highlight the identity you are adding from the drop-down list. Add as many identities as you want, one at a time.

   For an application, click the Add button and highlight the identity you are adding from the drop-down list. Add as many identities as you want, one at a time.

Use the Remove button to remove a trusted identity.

You can check the settings of your trusted identities from the Advanced tab by viewing the `com.sybase.jaguar.server.trustedidentities` property, and the `com.sybase.jaguar.server.applicaiton.trustedidentities` property. Do not set trusted identities in the Advanced tab since these values are overwritten by the values set in the Security tab.

Authentication of component invocation from servlets

This section describes how to propagate servlet credentials between servers and how to use identities to map logical identity names to a user name, password, and required SSL session characteristics. The identity names are used in the run-as mode settings on Beans called from a servlets.

Caller propagation for servlets on remote servers

A servlet's or JSP's HTTP client credentials are propagated when EJBs are invoked on remote servers. Earlier versions of EAServer propagated user name/password or digital IDs, and only within the same server.

Run-as support

Run-as support for servlets is similar to run-as support for EJBs:

- Run-as support is defined on a per-servlet basis.
- Run-as support applies to all method invocations on Beans called from the servlet.

To enable run-as support for servlets or JSPs, the servlet or JSP must be installed in a Web application. Additionally, you must specify `corbaname` URLs in the EJB Reference properties for the servlet or JSP that issues the call. For information on interoperable naming URLs, see Chapter 9, "EAServer EJB Interoperability," in the *EAServer Programmer's Guide*.

From EAServer Manager, configure the run-as identity as follows:

1 Expand the icon for your Web application.

2 Highlight the servlet for which you are configuring run-as support.

3 Display the Run As Identity tab and configure the settings as follows:

   - Run as – choose `specified`.
   - Role – specify a role name. The identity specified in the Mapped to Jaguar identity field should be in this role. This name is used if the Web application exported to a WAR file.
   - Run as identity – specify a logical identity name.
   - Mapped to Jaguar identity – choose an EAServer identity from the drop-down list. This is the identity with which the servlet invokes components.
   - Description – enter an optional text comment. This field can be used to provide identity mapping instructions for the deployer when the Web application is deployed to another server.

   The Existing Mappings on the Web Application table displays logical identity names that are mapped to EAServer identities by servlets in the same Web application.

You can configure a run-as identity application or server-wide. This provides a convenient way to globally set the run-as identity for all of the servlets/JSPs in an application or server. See "Configuring EJB 2.0 components or servlets to run as a different identity at the application or server level" on page 14 for more information.

Quality of protection

EAServer Manager allows you to set the quality of protection (QOP) for EAServer packages, components, and methods. QOP establishes a minimum level of encryption and authentication that a client must meet before it can access your business logic. For example, if you do not set a QOP at the package level, all clients can access the package. You can then set a QOP that restricts access to components within that package, and a different QOP that further restricts access to methods within those components.

This chapter discusses setting server-side QOP. For information about configuring client-side QOP, see:

- Chapter 5, “Using SSL in Java Clients”
- Chapter 6, “Using SSL in C++ Clients”
- Chapter 8, “Using SSL in ActiveX Clients”

Note The component’s QOP setting is ignored if the user is the system user; in other words, the user is jagadmin or the component is being called by a service or other component that runs with the system identity.

Naming service support

The client’s QOP, EAServer listener’s security profile, and the package, component, and method QOP work together to establish end-to-end security. To accommodate naming services and reduce connection time, a special CORBA component tag is set in the interoperable object reference (IOR). The naming service sends only profiles with QOPs that match a client’s QOP so that the client tries to access only listeners and packages, components, and methods for which the client has a compatible QOP.

Usage scenarios

Table 2-1 provides a hierarchy of QOP settings. For a given client to access your business logic:

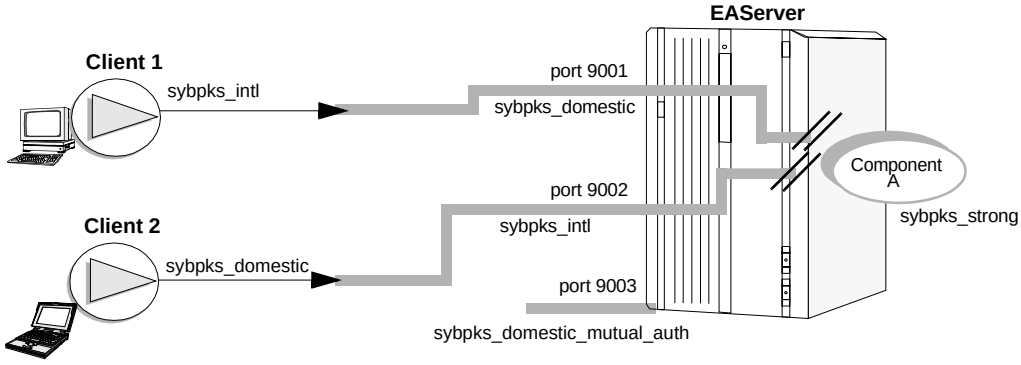
- A QOP-compatible listener must be available on the server, and
- Either the same or weaker QOP or no QOP restrictions must be placed on the package/component/method.

Table 2-1: QOP hierarchy

QOP hierarchy from weaker to stronger	Comments
syb_osauth sybpks_domestic_anon sybpks_simple sybpks_simple_mutual_auth sybpks_intl sybpks_intl_mutual_auth sybpks_domestic sybpks_domestic_mutual_auth sybpks_strong sybpks_strong_mutual_auth	Some QOP profiles overlap. For example, sybpks_domestic supports both 128-bit encryption and 40-bit encryption. If you use sybpks_domestic as a package QOP, a client QOP of sybpks_intl meets the minimum requirement of 40-bit encryption. sybpks_strong supports only 128-bit encryption and is compatible with only one of the domestic or strong profiles. For a list of CipherSuites supported by each QOP profile, see Table 12-2 on page 142.

Figure 2-1 illustrates two clients trying to access component A. A QOP of sybpks_strong is set for the component. To access the component, the client must use a QOP that meets the minimum requirements of the component's QOP, and communicate with a listener that also meets the minimum requirements of the component's QOP.

Figure 2-1: QOP usage



In Figure 2-1:

- Client 1 accesses the server at listener port 9001, but cannot access the component because the client's QOP does not meet the minimum requirements of component A.

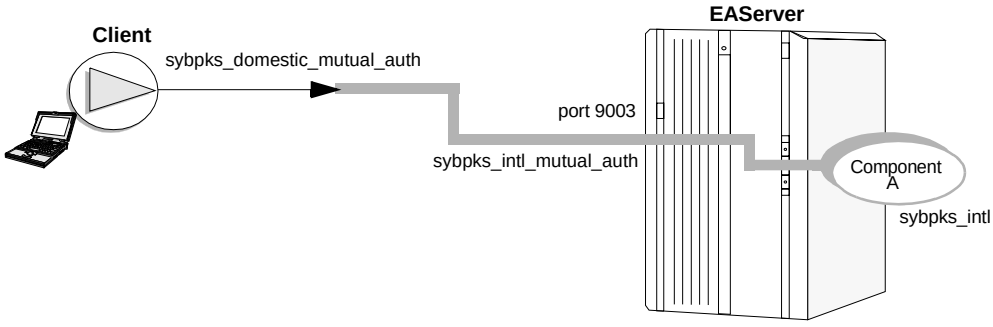
- Client 2 accesses the server at listener port 9002. The listener and client negotiate a cipher suite that both support. The highest cipher suite that both client and listener support uses 40-bit encryption and does not meet the minimum requirement of component A, since sybpks_strong supports only 128-bit encryption. Even though the client supports the minimum QOP required to communicate with component A, it is blocked because the listener does not support this minimum requirement.

  See Table 2-1 on page 19 and Table 12-2 on page 142 for more information about QOP compatibility.

- Neither client supports mutual authentication; consequently, neither can access the listener at port 9003.

  If a client has a QOP that includes mutual authentication, it can access a package, component, or method that does not, as long as there is a listener available to authenticate the client and the client's QOP meets the minimum level of security established at the package, component, or method. Figure 2-2 illustrates this scenario.

Figure 2-2: QOP-compatible listener



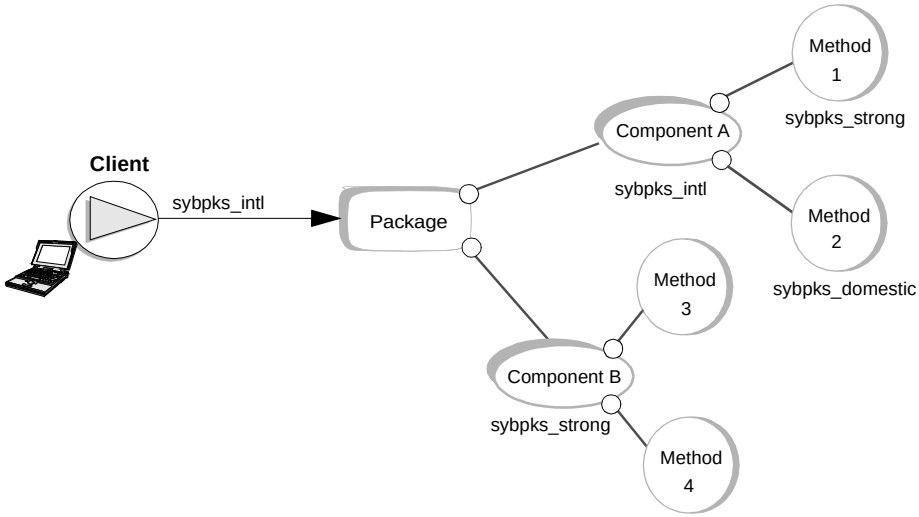
Controlling access to methods

Assuming that a compatible listener is configured on the server, Figure 2-3 illustrates a situation in which the client:

- Cannot access method 1 because the client's QOP does not match the minimum required by the method.
- Can access method 2 because sybpks_intl meets the security requirements of the method and component A, and the package has no QOP restrictions.
- Cannot access method 3 or 4 because it is blocked at the component level.

Setting a weaker QOP at the method than the component serves no purpose since the client will already be blocked at the component.

Figure 2-3: Using QOP to limit access to methods



syb_osauth

In addition to setting a QOP that establishes minimum encryption requirements, Jaguar provides another QOP, syb_osauth, for operating system authentication. You can set two QOP settings at the package, component, or method level, as long as one of them is syb_osauth:

- If syb_osauth is requested by the client and is not present in the package, component, or method QOP, the client ORB returns COMM_FAILURE and the message "no suitable profiles found."
- If the client does not request syb_osauth and the component, method, or listener QOP requires OS authentication, it is considered compatible (for backward compatibility with Jaguar 3.x and 2.0 clients). In this case, the user name and password are used for OS authentication.

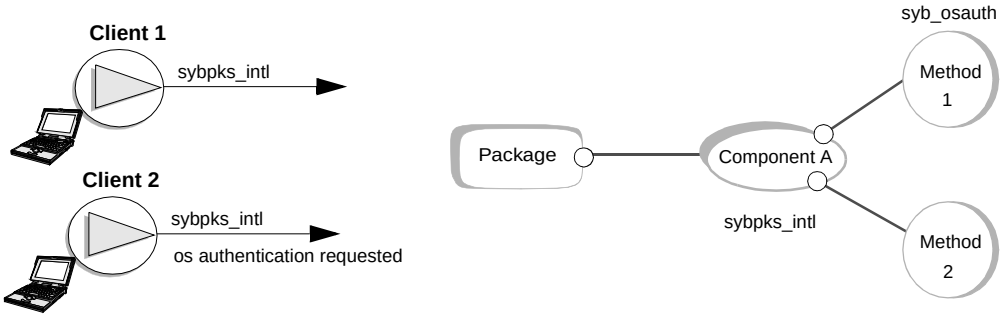
Note For syb_osauth to work properly, you must enable operating-system-based authentication server-wide (not at the listener level). If you do not, you cannot load packages, components, or methods that have the syb_osauth QOP set. See "Configuring OS authentication" on page 137 for information about enabling authentication for your operating system.

In Figure 2-4:

- Client 1 has a compatible QOP and supplies a user name and password to access method 1. Client 1 can access method 2 without authentication.

- Client 2 has a compatible QOP and uses authentication to access method 1 but gets a COMM_FAILURE error if it tries to access method 2.

Figure 2-4: Using syb_osauth



❖ **Configuring QOP from EAServer Manager**

Highlight the package, component, or method for which you want to establish a QOP.

1 Select File | Package, Component, or Method Properties.

2 Select the Advanced tab and set:

- The `com.sybase.package.qop` property for a package.
- The `com.sybase.component.qop` property for a component.
- The `com.sybase.method.qop` property for a method.

3 If the property already exists, you can highlight it and click Modify. Otherwise, click Add.

4 Enter the appropriate property name in the Property Name field and one (or two if using syb_osauth) of the values from Table 2-1 in the Property Value field.

After configuring QOP, you must either refresh or restart the server for your changes to take effect.

Client authorization

EAServer provides component authorization through both roles and custom components:

Roles EAServer's authorization model is based on roles. Define roles in EAServer Manager. Each role can include and exclude specific user names and digital certificates. If you use native operating system authentication, you can also include and exclude operating system group names; all users in the specified group are affected.

Roles are attached to EAServer packages, components, and methods. Attaching a role to a package controls access to all components in the package. To use a component, a user must be allowed component access by both the roles that are attached to the component and the roles that are attached to the package that contains the component.

See "Configuring EAServer roles" on page 129 for more information on defining roles.

Custom components EAServer provides role and authorization service components with which you can create and install your own component to authorize clients to access resources on EAServer. See Chapter 9, "Creating and Using Custom Security Components."

Enterprise JavaBeans

EJB 1.0 components use the package, component, and method role-based access control model used by all other component types. "Configuring EAServer roles" on page 129 describes how to configure roles and associate them with packages, components, and methods.

EJB 2.0 and 1.1 component security uses method-level constraints rather than the package and component role constraints used for other component models. The Roles folder does not display for EJB 2.0 or 1.1 components, or for packages that contain only EJB 2.0 or 1.1 components. If EJB 2.0 or 1.1 components are installed in a package that contains other component types, the package role folder has no effect on the EJB 2.0 or 1.1 components.

To restrict access beyond the configured permissions, you can call the isCallerInRole Java method to check the user's role membership. If you call the isCallerInRole Java method, you must configure role references to map names used in isCallerInRole calls to J2EE role names that are configured in the package properties.

❖ **Configuring logical role mappings**

Role settings in EJB 2.0 and 1.1 method permission use logical J2EE role names which must be mapped to EAServer role names in the properties of the package where the component is installed. The logical names are used when exporting the component to an EJB-JAR file. Configure role mappings as follows:

1 If necessary, define new EAServer roles to be used in the method level constraints. See “Configuring EAServer roles” on page 129 for details.

2 Display the package properties.

3 Display the Role Mapping tab and configure the mappings as follows:

- To add a logical J2EE role name, click Add and enter the role name.
- To specify the mapped EAServer role, click in the right column, opposite the J2EE role to be mapped, then use the pull-down menu to choose the mapped role.

❖ **Configuring EJB 2.0 or 1.1 method permissions**

1 If necessary, define new EAServer roles to be used by callers of the component and map them to J2EE roles in the package properties. You must map a J2EE role name for each role to be used in method permissions.

2 For each method that requires limited access, display the Method Properties dialog and highlight the Permissions tab. A check box displays for each mapped J2EE role in the package that contains the component. Select the check box by each role that can call the method.

Several predefined roles are available. For example, select the predefined “everybody” role if all users are to have access, or select “nobody” if all users are to be denied access. “Predefined roles” on page 135 describes the other predefined EAServer roles.

If the logical J2EE role is not mapped to an EAServer role, then at runtime, the server defaults to performing role checks against the logical J2EE role. The server assumes there exists an EAServer role with the same name as that of the logical J2EE role.

❖ **Configuring EJB 2.0 or 1.1 role references**

1 If necessary, define new EAServer roles to be used by callers of the component. and map them to J2EE roles in the package properties. You must map a J2EE role name for each role to be used in role references.

2 For each component that calls the isCallerInRole method, display the Component Properties dialog and highlight the Role Refs tab. Add or modify roles as follows:

- To add a role, click Add and edit the new entry as described below.
- To modify a role, edit the Reference Name (used in isCallerInRole calls), and choose the mapped J2EE role (configured in the properties of the package where the component is installed).

EJB 2.0 authorization of methods with no defined permissions

For EJB 2.0 components, if there are no roles associated with a method, then access is denied to everyone, which is different from EJB version 1.*x*. Keep this in mind when upgrading from EJB version 1.*x* to 2.0.

Note The com.sybase.jaguar.server.ejb.role.default property affects only EJB 2.0 components, not EJB 1.1 or 1.0 components.

Usually a role reference is mapped to a J2EE role. However, sometimes the mapping to the J2EE role may be missing, either intentionally, or due to mappings or customization not being performed after you deploy a J2EE application. In some cases, the mapping of the J2EE role to the EAServer role is missing. The following describes the behavior of the server in such cases:

1 If a role reference to J2EE role mapping is located, then a mapping between the J2EE role and an EAServer role is searched. If the search fails, role checks are performed against the J2EE role directly. If the search succeeds, then role checks are performed against the EAServer role.

2 If the role reference to J2EE role mapping search fails, then a mapping between the role reference and an EAServer role is performed. If a match is found, role checks are performed against the EAServer role. Otherwise, role checks are performed against the role reference directly.

In some EAServer and application configurations, the role reference name, the J2EE role name, and EAServer role name may be the same. In such cases, even though the mappings have not been explicitly set by the deployer at run time the server uses the default behavior, and EAServer performs the role checks internally against the EAServer role. In some application environments, this may be the intended and desired behavior, while in other environments, this may be unintended.

Role checks performed against a role that is not defined in the repository fail. If there is a role service or an authorization service, these services are consulted. See Chapter 9, “Creating and Using Custom Security Components.”

EJB 2.0 authorization For an EJB 2.0 bean, if there are no method permissions defined for all methods, no authorization checks are performed, and access is granted to any user. If however, any one method has a permission assigned to it, then you must assign permissions to all methods to allow client access, otherwise, your clients will be denied access due to an authorization failure.

CHAPTER 3

Using Web Application Security

This chapter discusses how to establish authentication and authorization levels for your Web application elements using declarative security provided by EAServer Manager.

Introduction

A Web container holds your Web application elements, including components, servlets, JSPs, HTML pages, and so on. The Web application's deployment descriptor describes how a Web application is deployed, including the level of security for the various elements of your Web application. For example, your Web application may include an HTML page that is available to all visitors to your site, while other HTML pages, servlets, and JSPs are restricted to existing or preferred customers.

Web client security requires that Web content be deployed in Web applications:

- There is no way to secure files deployed in EAServer's HTML root directory.
- Do not put sensitive information such as passwords in files that can be downloaded by Web clients.
- Do not put files containing sensitive information in locations that allow download by Web clients.

❖ **Accessing the security properties of your Web application from EAServer Manager**

1 Highlight the Web Applications or the Installed Web Applications folder.

2 Highlight the Web application for which you are establishing security.

3 Select File | Properties.

4 Select the Security tab from the Web Applications Property window.

You can now define the authentication method of your Web application and security constraints on the various elements within your Web application.

❖ **Defining Web application security from the Web application Security wizard**

As an alternative to setting Web application security from the Web Application Properties dialog, you can use the Web Application Security wizard, which guides you through the security configuration process.

1 Highlight the Web Applications or the Installed Web Applications folder.

2 Highlight the Web application for which you are establishing security.

3 Select File | Security Configuration Wizard.

4 Follow the instructions in the wizard to define the authentication method of your Web application and security constraints on the various elements within your Web application.

Authentication

The types of Web application authentication methods available include:

- **None** no authentication is required.
- **Basic** the server asks the client for its user name and password. You also provide a Realm name. The realm adds additional information to the client who is logging in to your site. For example, if you do not provide a realm name when a client visits your site, the browser displays a message to the client that states "The server at *host*:*port* wants you to log in." If you enter a realm name of "Human Resources Web site," the browser displays "The server at Human Resources Web site at *host*:*port* wants you to log in."

  When an HTTP client sends the HTTP basic authentication header:

- The server authenticates the client using the server-defined authentication scheme and invokes any defined customized authentication component.
- If the request is intended for PowerDynamo, the server still authenticates the client, and if the request is denied, HTTP status code 401 (Unauthorized) is sent back to the client.
- If the authentication fails, the request fails and an error message is sent back to the client. If the request is intended for a Web application, the Web application manages error handling.
- If the request is intended for a regular static page, the request is denied, and HTTP status code 401 (Unauthorized) is sent back to the client.

- **Form** the Web application developer creates an HTML login page, where the client enters a user name and password. The entire HTML page is sent to the server. You also create an error page that is returned to the client in the event of a server error.
  - Login page – enter the location of the login page that is supplied to the client at login. For example, */login.jsp* might be your login page.
  - Error page – enter the location of the error page that the client is directed to should a server error occur during login. For example, */error.jsp* might be your error page.

  Login and error pages can vary from a very simple HTML page to a complex page that includes servlets and JSPs.

  The location of the error and login pages is relative to the *WebApp* directory whether or not a "/" is used. For example, if you specify */error.jsp* or *error.jsp* as the location of your error page, the servlet engine assumes that it is contained in the *WebApp* context.

  Below is an example of a form login and error page. The action of the form login page must always be j_security_check. The user name and password fields should be j_username and j_password respectively.

  Form login page:

```
<html>
<body>
<h1>Login page</h1>

<form method="POST" action="j_security_check" >
<input type="text" name="j_username">
<input type="password" name="j_password">
```

```
<input type="submit" name="j_security_check">
</form>

</body>
</html>
```

Form error page:

```
<html>
<head>
<title>Login Error</title>
</head>
<body> Login error -- please try <a
href="login.html">again</a>.
</body>
</html>
```

These examples assume that *login.html* is the login page, and that the error page and login page are in the same directory.

- Client-cert – the client connects to the server using SSL tunneled within HTTP. The client must provide a certificate that the server accepts and authenticates. For more information about SSL, see Chapter 12, “Security Configuration Tasks,” and Chapter 13, “Managing Keys and Certificates.”

Note You cannot use both “client-cert” and “OS authentication” as Web application security mechanisms at the same time. If you do, clients will not connect to the Web application. See “Configuring OS authentication” on page 137.

Note EAServer does not support HTTP digest authentication. If you specify digest authentication, the default, Basic, is used instead.

EAServer supports *lazy authentication*, which means that the server attempts to identify a client only when the client attempts to access a restricted resource. As long as the client accesses only resources that do not require authorization, the server does not attempt to authenticate the client.

When a server authenticates a client, the client is authenticated for all applications and references on the server. You can implement authentication of a client for an entire server by using cookies or rewriting the URL. A reference to the client's security credentials is saved in a cookie or encoded in the URL.

Form login requirements in a Web application when using HTTPS (SSL)

To use the form login mechanism in your Web application, the client must support cookies. The client can be a browser or a standalone HTTP client. To convert your Web application, which uses the form login mechanism in conjunction with HTTPS, the transport guarantee for the form login page and the pages that require authorization must be identical. Otherwise, the client will receive multiple HTTP redirects to the same page, resulting in an error. See "Defining a security constraint from the Web Application Properties Security tab" on page 33 for information about configuring transport guarantee.

Here are the steps required to enable HTTPS for the eStore application, which is a large, comprehensive sample application developed by Sun Microsystems to run on J2EE-compliant servers. eStore simulates an online pet store implemented with Java Server Pages, Java servlets, and Enterprise Java Beans. You can download eStore as part of the Sun Microsystems J2EE Blueprints at http://java.sun.com/j2ee/blueprints/.

1 Change the transport guarantee for the existing two security constraints from None to Confidentiality or Integrity.

2 Add a new security constraint. Set the transport guarantee for the new security constraint to the same value as the existing two security constraints.

3 Add a Web resource collection to the new security constraint. Define a Web resource, and set the URL pattern to "*/login.jsp*", which is the URL of the form login page.

4 Refresh the eStore application. Connect to the eStore application from your browser. The form login and subsequent communication occurs using HTTPS.

Web application direct form login

EAServer supports direct form login, which allows you to access a Web application's protected content directly without requiring the user to visit the Web application's form login page.

To enable direct form login, set the following session property:

```
com.sybase.jaguar.servlet.session.redirecturl
```

This property specifies the URL of the protected page that you want to access. With the property set, submit a post request to the form login URL with the user name and password specified in the request parameters. If the login succeeds, EAServer redirects the user to the specified page.

If you do not specify a page to redirect to before posting a request to the login form, EAServer redirects the user to the page specified by this Web application property:

```
com.sybase.jaguar.webapplication.default.protectedpage
```

If this property is not set, EAServer redirects the user to the Web application's welcome page.

Also, when authentication fails, the following properties are set in the servlet session before invoking the error page:

- `com.sybase.jaguar.servlet.session.username` – the user name specified in the failed login attempt.
- `com.sybase.jaguar.servlet.session.password` – the password specified in the failed login attempt.

These settings are removed when authentication succeeds.

Authorization

Security constraints enable you to set various levels of authorization within the elements of your Web application. You create J2EE roles and map them to EAServer roles, then limit access to JSPs, servlets, and HTML pages to entities that belong to an authorized J2EE role. In addition, you can define which HTTP methods have access to which URLs, and establish levels of transport guarantee.

For example, you could create a security constraint that blocks access to all users at the Web application level. You could then grant access to resources (HTML pages, JSPs, servlets) within the Web application to authorized users. To do this, you need at least two security constraints:

1 Create a top-level security constraint and assign to it a Web resource collection with a URL pattern set to "/*".

   Establish an authorized role for the security constraint that contains no users. For example, you could create the role of "None" and assign it to the security constraint.

2 Create another security constraint and assign to it a Web resource collection with a URL pattern set to the URL locations for which you are providing access.

   Establish an authorized role that contains the users that are allowed access to the Web resources protected by this security constraint.

3 Create additional security constraints and allow access to other Web resources as needed.

Use this same approach to define security constraints that require specific levels of transport guarantee.

❖ **Defining a security constraint from the Web Application Properties Security tab**

1 Create a security constraint – click Add to create a security constraint. Security constraints are automatically named SC0, SC1, and so on.

   To delete a security constraint, highlight the constraint and click Delete.

2 Define a Web resource collection – Web resource collections contain a list of URL patterns and HTTP methods available for those URLs. To define a Web resource collection:

   a Highlight the security constraint to which the Web resource collection belongs, and click Edit.

   b Click Add to create a collection name. Provide a description.

   c Highlight the collection to which you are adding the Web resources you are protecting.

   d Add a URL pattern to be protected by clicking Add in the URL Patterns window.

   e Double-click "urlPattern" and enter the URL to be protected. Add additional URL patterns for this collection by repeating this step.

The URL pattern can have two forms:

- */url_name* – specifies an individual URL.
- */url_location/** – specifies all of the URLs located in the *url_location* directory.

f Select the HTTP operations that are allowed access to the defined URL patterns. HTTP operations include:

- GET – the most common method used by browsers. GET receives its input through a query string.
- POST – similar to a GET except that the input data is sent through standard input instead of using the query string. The POST method is normally used for an HTML form.
- PUT – same as POST except PUT usually implies that the operation take effect immediately whereas POSTs action may be delayed.
- OPTIONS – determines what HTTP options are supported.
- DELETE – removes some entity.
- TRACE – causes a response with a message containing all of the headers sent in the trace request.

3 Establish authorized roles – define the authorized roles that have access to the HTTP methods for the URLs defined for this security constraint. Before establishing an authorized role, you must map EAServer roles to J2EE roles. See “Role mapping” on page 36 for more information.

To configure role checking for a security constraint:

a Highlight the security constraint to which you are adding authorized roles.

b Select Enable Authorization if role checking is to be performed for this role. If this option is not selected, all users have access to pages associated with this security constraint.

Note To deny all users access to pages associated with the security constraint, select Enable Authorization, but do not assign any roles.

To allow access to all users, deselect the Enable Authorization option.

To force authentication to happen, but allow access to all users, create a J2EE role named 'everybody' and map it to the EAServer role of the same name. Pages that require this role will trigger authentication if the user is not already authenticated, and allow access to all users.

---

c Click the Authorized Roles Edit button.

d A list of mapped EAServer roles displays. Click the check box for the roles that have permission for the Web resources protected by this security constraint.

4 Transport guarantee – establish a level of transport security for each security constraint appropriate for the Web resources you are protecting. If you use basic or form-based authentication, passwords and other sensitive information is not protected for confidentiality. If you have sensitive information that you want to protect, establish a security constraint that uses a greater level of protection. Supported transport guarantee levels are:

- **None** uses insecure HTTP. Using SSL-protected sessions has more overhead than insecure HTTP sessions. Use None for transport guarantee if you do not need the added confidentiality of SSL.
- **Integral** uses an SSL-protected session that checks for data integrity.
- **Confidential** uses an SSL-protected session to ensure that all message content, including the client authenticators, are protected for confidentiality as well as data integrity. A Confidential transport guarantee has more overhead than None.

Role mapping

This section describes how to map EAServer roles to J2EE roles. Members of J2EE roles can be granted permission (authorized) to access Web resources protected by security constraints.

❖ **Mapping an EAServer role to a J2EE role**

1 Select the Role Mapping tab from the Web application properties window.

2 Click Add. Double-click the J2EE role and enter a name. You can also enter a description for the role in the provided field.

3 Select an EAServer role from the drop-down list. This is the role from which the J2EE role inherits its permissions and members.

See “Configuring EAServer roles” on page 129 for more information.

CHAPTER 4

Securing TDS Client Access

This chapter describes how Tabular DataStream (TDS) and Methods As Stored Procedures (MASP) clients access EAServer and the security features provided for these clients.

TDS and MASP listeners

TDS and MASP clients access EAServer through the TDS listener, which does not support SSL. The default EAServer TDS listener is located on port 7878.

See “Configuring listeners” on page 145 for information about establishing a TDS listener for your TDS and MASP clients. For added security, if you do not require a TDS listener, delete the associated listener from EAServer.

MASP client security

MASP clients can connect to EAServer and invoke components as if they were a Sybase database stored procedure. Authentication consists of operating system based authentication. Authorization consists of the same role-based authorization mechanism as used for IIOP clients.

See “Configuring OS authentication” on page 137 for information about authentication, and “Configuring EAServer roles” on page 129 for information about establishing role-based authorization. See Appendix A, “Executing Methods As Stored Procedures,” in the *EAServer Programmer's Guide* for more information about MASP clients.

Open Server client security

Open Server™ clients use the same security mechanisms when communicating with EAServer as regular Open Server applications except that EAServer does not support Kerberos or DCE. Open Server clients can also use EAServer supported OS based authentication. See “Configuring OS authentication” on page 137.

Open Server client security mechanisms include:

- **Login authentication services** The fundamental security service is *login authentication*, or confirming that users are who they say they are. Login authentication involves user names and passwords. Users identify themselves by their user name, then supply their passwords as proof of their identity.
- **Per-packet security services** You can protect your Open Server applications with a number of per-packet security services, including:
  - Data confidentiality – encrypts all transmitted data and assures that strangers cannot understand in-transit data.
  - Data integrity – detects attempts to tamper with in-transit data.
  - Data origin time stamping – assures that received data was really sent by the client or the server.
  - Replay detection – detects attempts by strangers to replay captured transmissions.
  - Sequence verification – detects transmissions that arrive in a different order than they were sent.
  - Channel binding – stamps each transmission with an encrypted description of the client’s and server’s addresses.

See the Open Client/Server documentation for detailed information about Open Server security.

For information about migrating your Open Server applications to EAServer, see Appendix B, “Migrating Open Server Applications to EAServer,” in the *EAServer Programmer’s Guide*.

CHAPTER 5

Using SSL in Java Clients

EAServer supports SSL connections from Java applets and applications. In deployment scenarios where clients connect to EAServer over the Internet, SSL can protect sensitive data transmitted over the network. For more information about SSL, see Chapter 12, “Security Configuration Tasks,” and Chapter 13, “Managing Keys and Certificates.”

Using SSL in Java applets

Java applet clients can use the Web browser’s Java SSL implementation to create SSL connections to an EAServer. SSL connections from a Java applet require that:

- You connect to a server listener that supports the desired level of security. You do this by specifying the address as an IOR string as described in “Creating a Manager instance” in Chapter 12, “Creating CORBA Java Clients,” of the *EAServer Programmer’s Guide*.
- Your Web browser recognizes and accepts the server listener’s SSL certificate.
- If using mutual authentication, you have a personal certificate installed in the Web browser’s certificate database, signed by a certificate authority that is recognized and trusted by the EAServer.

Chapter 15, “Tutorial: Using SSL” contains a security tutorial that walks you through the tasks of configuring certificates and running a sample applet that connects to EAServer using SSL.

Using SSL in Java applications

Java application clients create SSL connections using the same native implementation used by C++ and ActiveX clients.

Requirements

Make sure you select the *C++ Runtime* and *SSL Runtime* options when installing the EAServer client runtime. SSL support in Java applications requires the files installed by these options, including the *$JAGUAR_CLIENT_ROOT/DB* directory that contains the client-side security database which contains the certificates used to make SSL connections and the standalone Security Manager used to manage these certificates.

The client installation must also be correctly configured to load the native EAServer client libraries when you run your application. See the *EAServer Installation Guide* for more information. The following environment variable settings are required at runtime:

- JAGUAR_CLIENT_ROOT must specify the full path to the EAServer client runtime installation directory.
- On Windows platforms, PATH must include the EAServer client *dll* subdirectory.
- On UNIX platforms, the system's shared library search path (LD_LIBRARY_PATH on Solaris) must include the EAServer client *lib* subdirectory.
- CLASSPATH must include the EAServer client runtime classes. Specify the full path to the *easclient.jar* and *easj2ee.jar* files, or include the classes in these JAR files in a JAR that you build yourself.

Establishing a secure session

To ensure a secure session between your Java application and EAServer, you must configure SSL settings before using one of the standard techniques to instantiate proxies for the EAServer components.

You can configure the settings required for SSL connections using two techniques:

1. **By setting ORB properties** The required SSL settings must be known in advance, and your application can connect only to servers that use certificates issued by a known, trusted certificate authority.
2. **By using the SSLServiceProvider interface** The SSLServiceProvider interface allows your application to determine what options are available at runtime. In addition, you can supply a callback class with methods that supply settings as needed and respond to exceptional cases. For example, the client ORB invokes callback methods if the application specified an invalid certificate password or if a connection is made to a server that uses certificate issued by an unknown certificate authority.

Applications that run without user interaction typically configure SSL settings with the ORB properties. Interactive applications typically use the SSLServiceProvider interface and install a callback. When a callback is installed, you can rely on user interaction in the callback methods to configure necessary settings. For example, if the certificate password has not been supplied, the ORB invokes the getPin callback method.

Once you have correctly configured the required SSL settings, use the standard technique to instantiate proxies, as described in Chapter 12, "Creating CORBA Java Clients," in the *EAServer Programmer's Guide*. Proxies are created in a secure session as long as the server supports the requested level of security.

Using the SSLServiceProvider interface

The CtsSecurity.SSLServiceProvider interface provides setGlobalProperty and getGlobalProperty methods to set and retrieve the SSL properties listed in Table 5-1 on page 44. After initializing an ORB instance, you can instantiate a proxy for the SSLServiceProvider interface with the ORB.resolve_initial_references method, as shown below:

```
import CtsSecurity.*;

SSLServiceProvider sslServProv =
       SSLServiceProviderHelper.narrow
       (orb.resolve_initial_references
       ("SSLServiceProvider"));
```

You can then call the setGlobalProperty method to set properties, as in the example below:

```
prov.setGlobalProperty("qop", "sybpks_intl");
```

Properties set with the SSLServiceProvider interface affect all ORB instances used by the application. However, if an equivalent property has been set for an ORB instance, the ORB property value takes precedence.

You can retrieve property values using the getGlobalProperty method. For example:

```
String availQop[] = prov.getGlobalProperty("availableQop");
String qopDesc[] = prov.getGlobalProperty("availableQopDesc");
```

getGlobalPropertyMethod returns an array of strings. When retrieving properties that take a single value, the value is returned in an array of length 1.

These methods are also documented in the generated Interface Repository documentation for the CtsSecurity::SSLCallback interface. The generated documentation is linked to your EAServer's main HTML page.

SSL properties

Table 5-1 lists the ORB and SSLServiceProvider properties that govern the use of SSL. In addition, you need to connect to a server address that can support your chosen level of security, as described in "Secure server addresses" on page 47.

Some properties, if not set or set incorrectly, cause the ORB to invoke an SSL callback method. If you do not install an SSL callback, the default callback implementation aborts the connection attempt.

Table 5-1: SSL Properties

Property name for ORB.init	Property name for SSLServiceProvider	Description
com.sybase.CORBA.pin	pin	Always required when using SSL. Specifies the PKCS #11 token PIN. This is required for logging in to a PKCS #11 token for client authentication and for retrieving trust information. This property cannot be retrieved. If not set, set to "any", or set incorrectly, the ORB invokes the getPin callback method.

Property name for ORB.init	Property name for SSLServiceProvider	Description
com.sybase.CORBA.certificateLabel	certificateLabel	Required when using mutual authentication. Specifies the client certificate to use if the connection requires mutual authentication. The label is a simple name that identifies an X.509 certificate/private key in a PKCS #11 token. If the property is not set and the connection requires mutual authentication, the ORB invokes the getCertificateLabel callback method, passing an array of available certificate names as an input parameter.
com.sybase.CORBA.qop	qop	Always required when using SSL. Specifies the name of a security characteristic to use. See "Choosing a security characteristic" on page 46 for more information.
com.sybase.CORBA.userData	userData	Specifies user data (String datatype). This is an optional property. Client code can set user data during ORB initialization and access it using SSLSessionInfo::getProperty method in the SSL callback implementation. This may be useful as a mechanism to store ORB-level context information that is otherwise not available through the SSLSessionInfo interface.
com.sybase.CORBA.useEntrustID	useEntrustID	Specifies whether to use the Entrust ID or the Sybase PKCS #11 token for authentication. This is a Boolean (true or false) property. If this property is set to false, Sybase PKCS #11 token properties are valid and Entrust-specific properties are ignored. If this property is set to true, Entrust-specific properties are valid and Sybase PKCS #11 token properties are ignored.
com.sybase.CORBA.entrustUserProfile	entrustUserProfile	Specifies the full path to the file containing an Entrust user profile. This property is optional when the Entrust single-login feature is available and required when this feature is not available. If not set, the ORB invokes the getCredentialAttribute callback method.
com.sybase.CORBA.entrustPassword	entrustPassword	Specifies the password for logging in to Entrust with the specified user profile. This property is optional when the Entrust single-login feature is available and required when this feature is not available. If the password is required but not set, or set incorrectly, the ORB invokes the getPin callback method. This property cannot be retrieved.

Property name for ORB.init	Property name for SSLServiceProvider	Description
com.sybase.CORBA.entrustIniFile	entrustIniFile	Specifies the path name for the Entrust INI file that provides information on how to access Entrust. This is required when the useEntrustid property is set to true. If not set, the ORB invokes the getCredentialAttribute callback method.
none	callbackImpl	Name of a Java class that implements the CtsSecurity.SSLCallbackIntf interface. For example: `com.acme.AcmeSSLCallback` See "Implementing an SSL callback" on page 47 for more information.
none	availableQop	Retrieve only. A list of available security characteristics. The qop property can be set only to values that appear in this list.
none	availableQopDesc	Retrieve only. A list of descriptions for the available security characteristics, in the same order as listed in the value of the availableQop property.
none	entrustReady	Retrieve only. Returns true if Entrust PKI software is available on the client, false otherwise.

Choosing a security characteristic

To use SSL, you must specify a value for the qop property in ORB properties or by using the SSLServiceProvider interface. Specify the name of an available security characteristic. The characteristic describes the CipherSuites the client uses when negotiating an SSL connection. When connecting, the client sends the list of CipherSuites that it uses to the server, and the server selects a cipher suite from that list. The server chooses the first cipher suite in the list that it can use. If the server cannot use any of the available CipherSuites, the connection fails.

"Configuring security profiles" on page 139 describes the security characteristics that are provided with EAServer. At runtime, you can retrieve a list of characteristics and their descriptions by retrieving the availableQop and availableQopDesc properties.

Set the qop property to `sybpks_none` to prevent any use of SSL on a connection. This setting can be useful if you have set the property globally for all ORBs using the SSLServiceProvider interface, and you want to override the setting for an individual ORB instance.

Secure server addresses

The client ORB connects only to a server listener that uses an equivalent or greater level of security as requested in the qop setting. If you use the CosNaming or JNDI interfaces to instantiate proxies, the name service URL cannot specify a server address that uses a higher level of security than specified by the qop property. For example, if your server uses the typical port configuration, you can specify port 9000 (no SSL) in the name service URL if the qop specifies mutual authentication. However, you cannot specify port 9002 (mutual authentication) in the name service URL and set the qop to request server-only authentication. When you use ORB.string_to_object to instantiate a SessionManager::Manager proxy, the listener specified by the server address must use a security profile that matches the client's qop setting.

For more information on instantiating proxies, see Chapter 12, "Creating CORBA Java Clients," in the *EAServer Programmer's Guide*.

Implementing an SSL callback

An SSL callback class must implement the CtsSecurity.SSLCallbackIntf interface. The ORB invokes callback methods when required SSL settings have not been configured or a setting has an incorrect value. To install the callback, call SSLServiceProvider.setGlobalProperty to set the callbackImpl property, as in the example below:

```
sslprov.setGlobalProperty("callbackImpl",
     "Sample.ClientSSL.SSLCallbackExample.SSLCallbackExampleImpl");
```

The SSLCallbackIntf methods are as follows:

- **getCertificateLabel** Called when the session requires mutual authentication and a certificate label has not been provided in ORB properties or in SSLServiceProvider global properties. The callback receives an array of available certificate labels as an input parameter, and must return one of them or throw an exception to abort the connection attempt.
- **getCredentialAttribute** Called when additional information is required to use an Entrust certificate, such as the path to the Entrust profile file, or the path to the *entrust.ini* file.

- **getPin** Called when the certificate password has not been specified in ORB or SSLServiceProvider properties, or if the supplied password was incorrect. The implementation should check the "tokenName" property of the SSLSessionInfo instance to determine whether the requested password is for the Sybase certificate database or for an Entrust profile. Your implementation can throw an CtsSecurity.UserAbortedException to abort the connection attempt.
- **trustVerify** Called when the correct PIN for the certificate database has not been set, or if the server has presented a questionable certificate. The callback response determines whether the connection is allowed and, optionally, whether the certificate should be added to the local EAServer client certificate database.

Reason code	Description
CtsSecurity. REASON_TRUSTDBPINNOTSET. value	The password for the certificate database has not been set. Return CtsSecurity.TRUST_FAILED.value to cause the ORB to call the getPin callback method.
CtsSecurity. REASON_TRUSTDBLOGINFAILED. value	The password for the certificate database was incorrect. Return CtsSecurity.TRUST_FAILED.value to cause the ORB to call the getPin callback method.
CtsSecurity. REASON_UNKNOWN_CA. value	The root CA in the server's certificate chain is not listed in the Sybase certificate database.
CtsSecurity. REASON_CHAIN_EXPIRED. value	At least one certificate in the server's certificate chain has expired.
CtsSecurity. REASON_CHAIN_INCOMPLETE. value	Servers certificate chain is incomplete. The ORB cannot complete the chain using the CA certificates in the Sybase certificate database.

You must implement all of these methods in your class. If your implementation of a method does not process the request, throw an org.omg.CORBA.NO_IMPLEMENT exception so that the ORB uses the default response.

For more information about these callback methods, see the documentation for the CtsSecurity::SSLCallback interface in the generated Interface Repository documentation. "Sample Java applications that use SSL" on page 49 describes the SSL sample applications. These samples include an example SSL callback that interacts with the user.

Retrieving session security information

The CtsSecurity.SSLSession and CtsSecurity.SSLSessionInfo classes allow you to determine whether SSL is used on connections from a proxy to the server, and if so, retrieve the SSL session settings. The code below illustrates the sequence of calls:

```
... code to set ORB ssl properties, create session,
     instantiate proxy myComp ...
SSLSession sslSession =
  SSLSessionHelper.narrow(myComp);
try {
  SSLSessionInfo sslSessionInfo =
    sslSession.getSessionInfo();
} catch (CtsSecurity.SSLNotEnabledError e) {
     ... this means the proxy does not use SSL ...
}
```

You can call SSLSessionHelper.narrow to obtain the session information associated with any CORBA object.

The SSLSessionInfo methods allow you to determine the SSL session properties, such as the server's address, the client certificate in use, the server certificate in use, and so forth. For more information, see the Interface Repository documentation for the CtsSecurity::SSLSessionInfo interface. "Sample Java applications that use SSL" on page 49 describes the SSL sample applications. These examples show how to retrieve and print the SSL session information.

Sample Java applications that use SSL

Two sample Java applications that use SSL are in the *html/classes/Sample/ClientSSL* directory of your EAServer installation. The SSLDemo application allows you to configure SSL sessions using ORB properties set on the command line. The application uses the SSLSessionInfo interface to print a description of the SSL session. The SSLCbDemo application uses the SSLServiceProvider interface and an SSL callback class to query the user for SSL settings.

Creating HTTP and HTTPS connections in Java applications

You can create HTTP connections in Java applications using the HTTP protocol handling code built in to the Java Developer's Kit, and HTTPS connections using the HTTPS protocol handler provided with EAServer.

HTTP connections

The standard Java virtual machine provides HTTP connectivity with these classes in java.net package:

- URL allows you to use Uniform Resource Locator strings for HTTP connections and other protocol connections that can be represented by URLs.
- URLConnection represents a connection to a server and resource indicated by a URL.
- HttpURLConnection extends URL with additional methods that are specific to the HTTP protocol.

For details on these classes, see the JDK documentation. The following code shows a typical example. This code opens a connection, retrieves the data (text is assumed), and prints it:

```
URL url = new URL("http://www.sybase.com/");
URLConnection conn = url.openConnection();
conn.connect();
InputStreamReader content
    = new InputStreamReader(conn.getInputStream());
for (int i=0; i != -1; i = content.read())
{
    System.out.print((char) i);
}
```

HTTPS connections

The procedure for creating HTTPS connections is similar to that for HTTP connections, except that you must install EAServer's HTTPS protocol handler in the Java virtual machine and configure SSL parameters before opening a connection.

System requirements EAServer's HTTPS protocol handler uses the same SSL implementation as used by Java and C++ IIOP clients and requires a full client runtime install. For information on system requirements, see "Requirements" on page 42.

Installing the HTTPS protocol handler

The EAServer HTTPS protocol handler can be installed two ways:

- By configuring the `java.protocol.handler.pkgs` Java system property, making it the default handler for all HTTPS URLs. This is the recommended approach if you do not need to use another vendor's HTTPS protocol handler in addition to the EAServer implementation.
- By calling one of the java.net.URL constructors that takes a java.net.URLStreamHandler as a parameter. This approach must be used if you must use more than one HTTPS protocol handler in one EAServer or in one client application.

Configuring the default protocol handlers

The `java.protocol.handler.pkgs` Java system property configures the Java virtual machine default URL protocol handlers. To use the EAServer handlers, you must add com.sybase.jaguar.net to the list. For more information on this property, see the documentation for java.net.URL in JDK 1.2 at http://java.sun.com/products/jdk/1.2/docs/api/java/net/URL.html.

In a client application, specify this property on the command line; for example:

```
jre -Djava.protocol.handler.pkgs=com.sybase.jaguar.net ...
```

For an EAServer, set the JVM options property using the Advanced tab in the Server Properties dialog box:

Property	Value
com.sybase.jaguar.server.jvm.options	If not already set, set to: `-Djava.protocol.handler.pkgs=com.sybase.jaguar.net` If already set, verify that the value includes this option. JVM options must be separated with a comma.

You can specify more than one package by separating package names with a | (pipe) character, but you can configure only one handler per protocol.

Specifying protocol handlers at runtime

If you must use more than one HTTPS protocol handler in one EAServer or in one client application, you must call one of the java.net.URL constructors that takes a java.net.URLStreamHandler as a parameter. The specified java.net.URLStreamHandler instance overrides the default handler for the protocol specified by the URL. For example, to specify the EAServer HTTPS handler, use code like this:

```
import java.net.*;
import com.sybase.jaguar.net.JagURLStreamHandlerFactory;
import com.sybase.jaguar.net.HttpsURLConnection;

....

String url_string = "https://localhost:8081/index.html";

// The URL stream handler factory is required to create a stream
// handler.
JagURLStreamHandlerFactory fact = new JagURLStreamHandlerFactory();

// Extract the protocol from the front of the URL string
String protocol = url_string.substring(0, url_string.indexOf(":"));

// If the protocol is HTTPS, use the EAServer HTTPS handler. Otherwise,
// use the default handler
java.net.URL url;
if (protocol.equals("https"))
{
    url = new URL((URL)null, url_string,
        fact.createURLStreamHandler(protocol));
} else
{
    url = new URL(url_string);
}
```

EAServer's HttpsURLConnection class

EAServer provides the com.sybase.jaguar.net.HttpsURLConnection class to support HTTPS connectivity. This class extends java.net.URLConnection and implements all methods of java.net.HttpURLConnection. HttpsURLConnection provides these additional methods specifically for SSL support:

- A setSSLProperty method with signature:

```
void setSSLProperty (String prop, String value) throws
  CtsSecurity.InvalidPropertyException,
  CtsSecurity.InvalidValueException
```

Call this method to set the SSL properties described in “SSL properties” on page 55.

- A setSSLProperties method with signature:

```
void setSSLProperty (java.util.Properties props) throws
  CtsSecurity.InvalidPropertyException,
  CtsSecurity.InvalidValueException
```

This method is the same as setSSLProperty, but allows you to set multiple properties with one call.

- A getSSLProperty method with signature:

```
String[] setSSLProperty (String prop) throws
  CtsSecurity.InvalidPropertyException
```

Call this method to retrieve the SSL properties described in “SSL properties” on page 55.

- A setGlobalProperty method with signature:

```
void setGlobalProperty (String prop, String value) throws
  CtsSecurity.InvalidPropertyException,
  CtsSecurity.InvalidValueException
```

Call this method to set the global SSL properties described in “SSL properties” on page 55. Properties set with this method affect the handling of all HTTPS connections, not just the current one.

- A getGlobalProperty method with signature:

```
String[] getGlobalProperty(String prop) throws
  CtsSecurity.InvalidPropertyException;
```

Call this method to retrieve the global SSL properties described in “SSL properties” on page 55.

- A getSessionInfo method with signature:

```
CtsSecurity.SSLSessionInfo getSessionInfo() throws
CtsSecurity.SSLException
```

The SSLSessionInfo methods allow you to determine the SSL session properties, such as the server’s address, the client certificate in use, the server certificate in use, and so forth. For more information, see the Interface Repository documentation for the CtsSecurity::SSLSessionInfo IDL interface. getSessionInfo throws an a SSLException instance if SSL is not used on the connection.

❖ **Creating HTTPS connections**

1 Configure or install the EAServer HTTPS protocol handler as described in "Installing the HTTPS protocol handler" on page 51.

2 Create URL and URLConnection instances. If connecting to an EAServer, specify the address of an HTTPS listener that supports the desired level of security. For example:

```
URL url = new URL("https://myhost:8081/index.html");
URLConnection conn = url.openConnection();
```

3 Verify that the object returned by URL.openConnection is of class com.sybase.jaguar.net.HttpsURLConnection, then set SSL properties for the connection. "SSL properties" on page 55 describes the SSL properties that can be set. At a minimum, you must specify the qop and pin properties, as well as the certificateLabel property if using mutual authentication. For example:

```
if (conn instanceof HttpsURLConnection)
{
  HttpsURLConnection https_conn =
    (HttpsURLConnection) conn;
  try
  {
    https_conn.setSSLProperty( "qop","sybpks_intl"
);
    https_conn.setSSLProperty( "pin", "secret");
    https_conn.setSSLProperty(
           "certificateLabel", "John Smith");
  }
  catch ( CtsSecurity.InvalidPropertyException ipe )
  {
    System.err.println( ipe );
  }
  catch ( CtsSecurity.InvalidValueException ive )
  {
    System.err.println( ive );
  }
```

4 Open the connection, for example:

```
conn.connect();
```

Once the connection is open, you can perform any valid operation for a connection that uses java.net.HTTPUrlConnection. You can also call the getSessionInfo method to retrieve a CtsSecurity.SSLSessionInfo instance that allows you to verify the SSL connection parameters. For example:

```
java.net.URLConnection conn;
... deleted code that constructed URLConnection ...
if (conn instanceof HttpsURLConnection)
{
  HttpsURLConnection https_conn =
    (HttpsURLConnection) conn;
  CtsSecurity.SSLSessionInfo sessInfo =
    https_conn.getSessionInfo();
```

The SSLSessionInfo methods allow you to determine the SSL session properties, such as the server's address, the client certificate in use, the server certificate in use, and so forth. For more information, see the Interface Repository documentation for the CtsSecurity::SSLSessionInfo interface.

SSL properties

Table 5-2 lists the properties that can be set and retrieved with the HttpsURLConnection getSSLProperty, getGlobalProperty, setSSLProperty, and setGlobalProperty methods. Global properties are set and read with the getGlobalProperty and setGlobalProperty methods. Global properties affect all HTTPS connections, not just the HttpsUrlConnection instance on which they are set. The right column in Table 5-2 lists which methods are valid for each property.

Some properties, if not set or set incorrectly, cause the connection to invoke an SSL callback method. You can install a callback to respond to these cases with the callbackImpl global property. If you do not install an SSL callback, the default callback implementation aborts the connection attempt.

Table 5-2: HTTPS Properties

Property name	Description	Valid for methods
pin	Always required when using SSL. Specifies the PKCS #11 token PIN. This is required for logging in to a PKCS #11 token for client authentication and for retrieving trust information. This property cannot be retrieved. If not set, set to "any", or set incorrectly, the connection invokes the getPin callback method.	setSSLProperty setGlobalProperty

Property name	Description	Valid for methods
certificateLabel	Required when using mutual authentication. Specifies the client certificate to use if the connection requires mutual authentication. The label is a simple name that identifies an X.509 certificate/private key in a PKCS #11 token. If the property is not set and the connection requires mutual authentication, the connection invokes the getCertificateLabel callback method, passing an array of available certificate names as an input parameter.	setSSLProperty getSSLProperty setGlobalProperty getGlobalProperty
qop	Always required when using SSL. Specifies the name of a security characteristic to use. See "Choosing a security characteristic" on page 57 for more information.	setSSLProperty getSSLProperty setGlobalProperty getGlobalProperty
userData	Specifies user data (String datatype). This is an optional property. Client code can set user data during connection initialization and access it using SSLSessionInfo::getProperty method in the SSL callback implementation. This may be useful as a mechanism to store connection-level context information that is otherwise not available through the SSLSessionInfo interface.	setSSLProperty getSSLProperty setGlobalProperty getGlobalProperty
useEntrustID	Specifies whether to use the Entrust ID or the Sybase PKCS #11 token for authentication. This is a Boolean (true or false) property. If this property is set to false, Sybase PKCS #11 token properties are valid and Entrust-specific properties are ignored. If this property is set to true, Entrust-specific properties are valid and Sybase PKCS #11 token properties are ignored.	setSSLProperty getSSLProperty setGlobalProperty getGlobalProperty
entrustUserProfile	Specifies the full path to the file containing an Entrust user profile. This property is optional when the Entrust single-login feature is available and required when this feature is not available. If not set, the connection invokes the getCredentialAttribute callback method.	setSSLProperty getSSLProperty setGlobalProperty getGlobalProperty

Property name	Description	Valid for methods
entrustPassword	Specifies the password for logging in to Entrust with the specified user profile. This property is optional when the Entrust single-login feature is available and required when this feature is not available. If the password is required but not set or set incorrectly, the connection invokes the getPin callback method. This property cannot be retrieved.	setSSLProperty setGlobalProperty
entrustIniFile	Specifies the path name for the Entrust INI file that provides information on how to access Entrust. This is required when the useEntrustid property is set to true. If not set, the connection invokes the getCredentialAttribute callback method.	setSSLProperty getSSLProperty setGlobalProperty getGlobalProperty
callbackImpl	Specifies the name of a Java class that implements the CtsSecurity.SSLCallbackIntf interface. For example: `com.acme.AcmeSSLCallback` See "Implementing an SSL callback" on page 47 for more information.	setGlobalProperty getGlobalProperty
availableQop	Retrieve only. A list of available security characteristics. The qop property can be set only to values that appear in this list.	getGlobalProperty
availableQopDesc	Retrieve only. A list of descriptions for the available security characteristics, in the same order as listed in the value of the availableQop property.	getGlobalProperty
entrustReady	Retrieve only. Returns true if Entrust PKI software is available on the client, false otherwise.	getGlobalProperty

Choosing a security characteristic

To use SSL, you must specify the name of an available security characteristic as the value for the qop property. The characteristic describes the CipherSuites the client uses when negotiating an SSL connection. When connecting, the client sends the list of CipherSuites that it uses to the server, and the server selects a cipher suite from that list. The server chooses the first cipher suite in the list that it can use. If the server cannot use any of the available CipherSuites, the connection fails.

Chapter 12, “Security Configuration Tasks” describes the security characteristics that are provided with EAServer. At runtime, you can retrieve a list of characteristics and their descriptions by retrieving the availableQop and availableQopDesc properties.

Using Java Secure Socket Extension classes

The Java Secure Socket Extension (JSSE) is a set of Java packages that implements SSL and Transport Layer Security, which enables data encryption, server authentication, message integrity, and client authentication. JSSE is a client-side feature, which can be used with EAServer when it has been configured for SSL communication. For more information on SSL, see Chapter 13, “Managing Keys and Certificates.”.

Note JSSE does not contain any actual cryptographic logic. You must obtain an API package that performs the cryptographic functions, such as Bouncy Castle or Cryptix, which are available free over the Internet.

❖ **Setting up your JSSE environment**

1 Download and install the JSSE according to the documentation on the Java Web page at http://java.sun.com/products/jsse. The basic steps are:

- Copy the JSSE JAR files to the *jre/lib/ext* directory in your JDK installation.
- Edit the *jre/lib/security/java.security* file in your JDK installation, and add this line:

```
security.provider.2=com.sun.net.ssl.internal.ssl.Provider
```

From the Sun documentation, note the following:

- JSSE 1.0.2 requires JDK 1.2.2 or higher.
- JRE 1.3.1_02 includes the Java plug-in HTML Converter 1.3.1_02, which works best with JDK 1.3.
- JDK/JRE 1.3 or higher is recommended to run HTML applets.
- The Java plug-in HTML Converter is recommended for HTML applet clients.

2 Download and install the Java Plug-in HTML Converter, either version 1.3.1 or 1.4.

If you install version 1.3.1:

a Download and install JSSE 1.0.2 in the JDK 1.3.1 *jre/lib/ext* subdirectory of the JDK installation.

b Set up *jre/lib/security/java.security* according to the JSSE 1.0.2 directions.

3 The JSSE Samples Web page at http://java.sun.com/j2se/1.4/docs/guide/security/jsse/samples/index.html includes samples that create clients using JSSE. Verify that the samples compile and run with your JDK. You must be able to use the Java samples to request the secure VeriSign Web page at https://www.verisign.com.

4 Start EAServer and connect using EAServer Manager | Certificates folder.

5 In the User Certificates folder, highlight the Sample 1 Test ID certificate, and select File | Certificate Info. Confirm that the Sample1 Test ID certificate is valid; that is, that the current date falls between the certificate's Not Valid Before and Not Valid After dates.

6 From the User Certificates folder, export Sample1 Test ID as a Binary Encoded X509 Certificate (**.crt*). For example, save to a file named *eas.crt*.

7 Using the Java keytool, import the *eas.crt* file; for example:

```
keytool -import -file eas.crt -keystore $JAGUAR_JDK13/jre/lib/security/
[cacerts | jssecacerts] -trustcacerts
```

To simplify things, use the default certificate store cacerts; the password is "changeit".

8 To run a JSSE client application; for example, *ClientApp*:

a Create a *ClientApp.bat* file with these lines:

```
set classpath=%JAGUAR%\java\lib\easclient.jar; \
    %JAGUAR%\java\lib\easj2ee.jar;%classpath%
java -Djava.protocol.handler.pkgs=
   com.sun.net.ssl.internal.www.protocol ClientApp
```

b Run *ClientApp.bat*.

If you do not have a Web proxy, remove the Web proxy settings from your client, and enter the server information; for example:

```
iiops://localhost:9001, or
```

```
iiops://<host_name>:9001
```

Note The following steps apply only to HTML applets.

9 Remove these client ORB properties from your HTML applet client, if appropriate:

- com.sybase.CORBA.WebProxyHost=localhost
- com.sybase.CORBA.WebProxyPort=80
- com.sybase.CORBA.LogFile=.\iiop.log

10 To access your Web page from a Web browser, enter:

```
http://<host_name>:8080/jssehtml/yourAppClient.html
```

Where *yourAppClient.html* is your HTML applet client.

11 In the applet, enter `iiops://<host_name>:9001` as the connection parameter, and click Connect.

Note Sybase recommends using a Web browser that supports the Java Plug-in 1.3.1 or higher and the Java Plug-in Converter 1.3.1 or higher

Configuring ORB settings

Direct IIOP connections using JSSE are not supported.

❖ **Tunnelling IIOP through HTTPS (JSSE socket) using HTTP GET requests**

IIOP is contained within the HTTP packets.

1 Set the client URL to iiops://<*host_name*>:9001.

2 Set the following client ORB properties. To enable the EAServer message service to access the ORB properties, set the properties using the JMS property names; otherwise, use the CORBA property names.

JMS property	CORBA property	Vale
org.omg.CORBA.ORBClass	org.omg.CORBA.ORBClass	com.sybase.CORBA.ORB
com.sybase.jms.https	com.sybase.CORBA.https	true
com.sybase.jms.useJSSE	com.sybase.CORBA.useJSSE	true
com.sybase.jms.forceSSL	com.sybase.CORBA.forceSSL	true

❖ **Tunnelling IIOP through HTTPS (JSSE socket) using HTTP POST requests**

IIOP is contained within the HTTP packets.

1 Set the client URL to iiops://<*host_name*>:9001

2 Set the following client ORB properties. To enable the EAServer message service to access the ORB properties, set the properties using the JMS property names; otherwise, use the CORBA property names..

JMS property	CORBA property	Vale
org.omg.CORBA.ORBClass	org.omg.CORBA.ORBClass	com.sybase.CORBA.ORB
com.sybase.jms.https	com.sybase.CORBA.https	true
com.sybase.jms.useJSSE	com.sybase.CORBA.useJSSE	true
com.sybase.jms.forceSSL	com.sybase.CORBA.forceSSL	true
com.sybase.jms.HttpUsePost	com.sybase.CORBA.HttpUsePost	true

❖ **Tunnelling IIOP through an HTTPS connect (JSSE socket) using HTTP GET requests**

IIOP is contained within the HTTP packets.

1 Set the client URL to iiops://<*host_name*>:9001.

2 Set the following client ORB properties. To enable the EAServer message service to access the ORB properties, set the properties using the JMS property names; otherwise, use the CORBA property names.

JMS property	CORBA property	Vale
org.omg.CORBA.ORBClass	org.omg.CORBA.ORBClass	com.sybase.CORBA.ORB
com.sybase.jms.https	com.sybase.CORBA.https	true
com.sybase.jms.WebProxyHost	com.sybase.CORBA.WebProxyHost	<*web_proxy_host_name*>
com.sybase.jms.WebProxyPort	com.sybase.CORBA.WebProxyPort	<*web_proxy_port*>
com.sybase.jms.useJSSE	com.sybase.CORBA.useJSSE	true
com.sybase.jms.forceSSL	com.sybase.CORBA.forceSSL	true

❖ **Tunnelling IIOP through an HTTPS connect (JSSE socket) using HTTP POST requests**

IIOP is contained within the HTTP packets.

1 Set the client URL to `iiops://<host_name>:9001.`

2 Set the following client ORB properties. To enable the EAServer message service to access the ORB properties, set the properties using the JMS property names; otherwise, use the CORBA property names.

JMS property	CORBA property	Vale
org.omg.CORBA.ORBClass	org.omg.CORBA.ORBClass	com.sybase.CORBA.ORB
com.sybase.jms.https	com.sybase.CORBA.https	true

JMS property	CORBA property	Vale
com.sybase.jms.HttpUsePost	com.sybase.CORBA.HttpUsePost	true
com.sybase.jms.WebProxyHost	com.sybase.CORBA.WebProxyHost	*<web_proxy_host_name>*
com.sybase.jms.WebProxyPort	com.sybase.CORBA.WebProxyPort	*<web_proxy_port>*
com.sybase.jms.useJSSE	com.sybase.CORBA.useJSSE	true
com.sybase.jms.forceSSL	com.sybase.CORBA.forceSSL	true

Note The first time you connect may take a while because JSSE goes through an SSL authentication process.

Using an unsigned JAR

When using an unsigned JAR, your code runs with the default EAServer Manager | Certificates folder, which is fairly restrictive. To improve performance, you can edit Java's default security policy file using the instructions in Sun's security documentation. To enable EAServer's ORB to work in an unsigned environment:

- You must grant the ORB permission to read the proxy host settings, using one of these methods:

```
permission java.util.PropertyPermission "*", "read"
```

or

```
permission java.util.PropertyPermission "javaplugin.proxy.config.*",
"read"
```

- The ORB may require socket connect permissions to connect to a proxy server.
- If you are using the sample test certificate generated by EAServer, the EAServer certificate authority must be installed. You can do this in either the *cacerts* or the *jssecacerts* keystore using this syntax:

  keytool -import -file <*file_name*> -keystore [cacerts | jssecacerts]

  The password for the cacerts keystore is "changeit".

Note With a signed applet, you do not need to set permissions at the plug-in level. A signed JAR file describes the type of permissions it requires.

Sample security file

You can find a sample JDK security file in the JDK installation, in file *jre/lib/security/java.security*.

Possible solutions for JSEE issues

Cannot load applet

If you cannot load an HTML applet from your Web browser:

1 In the Tools | Internet Options dialog box:

- On the Connections tab, select Settings, then deselect Use a Proxy Server. Or, if you use a proxy server, verify the information is valid.
- On the Advanced tab, under Browsing, select Browse in a New Process.

2 In the Control Panel, double-click Java Plug-in 1.3.1_02. In the Java Plug-in Control Panel:

- On the Basic tab, select Enable Java plug-in and Show Java Console.
- On the Proxies tab, either select Use Browser Settings, or verify the Proxy Settings.
- Verify settings on the other tabs.

3 Shut down all Web browser sessions.

4 Close all Java console sessions; for example, from the Java Plug-in.

5 Restart your Web browser.

6 Delete all your temporary and cache files.

7 Reload the HTML applet page.

Debugging

If necessary, use the Java Plug-in console for debugging; set to debug level 5. If you reset the debug level, refresh the HTML applet.

CHAPTER 6

Using SSL in C++ Clients

Introduction

A C++ client can use IIOP tunnelled within SSL (also called IIOPS) to establish a secure session with EAServer.

Note For more information about security, including issuing certificates, see Chapter 13, “Managing Keys and Certificates.”

To establish a secure session with EAServer, follow these steps:

Step	What it does	Detailed explanation
1	Initialize the SSL security service as an ORB.	“Initializing the SSL security service” on page 66
2	Initialize the client ORB and create an *ORB* reference.	“ORB properties for secure sessions” on page 67
3	Use the *ORB* reference to create a *Manager* instance for the server.	“ORB properties for secure sessions” on page 67
4	Use the *Session* instance to create stub component instances. This step is the same regardless of whether the application uses SSL.	Chapter 15, “Creating CORBA C++ Clients,” in the *EAServer Programmer’s Guide*
5	Optionally, you can retrieve security information about the session.	“Retrieving session security information” on page 69

An example that illustrates all of these steps is in the *sample/ClientSSL* subdirectory of your EAServer installation.

Initializing the SSL security service

To initialize the SSL security service, you must retrieve the SSL security service context and set the quality of security services as well as any global properties for that context.

You must decide if you want to:

- Respond to any authentication request by the server.
- Use the Sybase PKCS #11 token (the default) or an Entrust ID.

Retrieve the SSL security service context

In this example, you use CORBA::ORB_init to initialize the ORB as an instance, *orb1*.

```
CORBA::ORB_var orb1 =
CORBA::ORB_init(argc,argv, "");
```

Use resolve_initial_references to obtain the initial context from the SSL security service URL string (`SSLServiceProvider`) as an object reference, *object*, on *orb1*. You must use `SSLServiceProvider` as the URL string. You use CtsSecurity::SSLServiceProvider::_narrow to convert *object* to the *sslServProv* instance (an instance of the SSLServiceProvider interface).

```
object = orb1->resolve_initial_references
    ("SSLServiceProvider");
sslServProv = CtsSecurity::SSLServiceProvider
    ::_narrow(object);
```

Set the quality of security services and global properties

To return the available qualities of security services from the availableQop property, call getGlobalProperty on the sslServProv instance. The qualities of security services refer to the security profile characteristic, which specifies the supported CipherSuites.

```
// query Available quality of services and set
// whatever we want.
CtsSecurity::StringSeq_var * availQop =
  sslServProv->getGlobalProperty("availableQop");
```

At this time, you can also set any global properties, such as the callback component with the callbackImpl property. You specify the callback component using the setGlobalProperty method. The setGlobalProperty method takes the name of the global property, callbackImpl, and the name of the callback component. The name of the component is the DLL or shared library name (without the file extension) followed by a forward slash, and the package and component name separated by forward slashes as shown in this example:

```
// Set callbacks.

sslServProv->setGlobalProperty
    ("callbackImpl", "myDLL/myPackage/myComponent");
```

Enable client authentication

To respond to a server's request for client authentication, you can:

- Use the setGlobalProperty method to set the *certificateLabel* property to the client certificate to use when the server asks for one, or
- Use the *callback* interface to provide a dialog (GUI- or text-based) where the user can enter a certificate to be sent back to the server.

ORB properties for secure sessions

You must set the ORBqop property when initializing the client ORB in order to use one of the available security profile characteristics. The security profile characteristic lists the CipherSuites the client uses when negotiating an SSL connection. The client sends the list of CipherSuites that it uses to the server, and the server selects a cipher suite from that list. The server must choose the first cipher suite in the list that it can use.

In this example, the ORBqop property is specified as sybpks_strong (strong 128-bit encryption) and the ORBuserdata property is specified as `myUserData`. The CORBA::ORB_init method initializes the client ORB (*orb2*) with these properties.

```
// Now configure a specific ORB instance,
// overriding the default Quality of
// service. Might want to connect to a server
// only using 128bit encryption.
Properties props(argc, argv);
props.put("ORBqop", "sybpks_strong");
props.put("ORBuserData", myUserData);
orb2 = CORBA::ORB_init(props.argc(),
      props.argv(), "");
```

You can also set these properties when initializing the client ORB:

- **ORBcertificateLabel** Specifies the client certificate to use, if the server requests mutual authentication. The label is a simple name that identifies an X.509 certificate/private key in a PKCS #11 token. You must set this property if the server will request the client's certificate. If this property is not set and the server requests client authentication, credentialCallback is invoked. If you set this property to "any", then the getCertificateLabel method in the SSLCallback interface is invoked. If client authentication is requested and neither the certificateLabel property nor the credentialCallback is set, the SSL session fails.
- **ORBpin** Specifies the PKCS #11 token PIN. This is required for logging in to a PKCS #11 token for client authentication and for retrieving trust information. If this property is not set and the server requests client authentication, the Login callback implementation is invoked to get the PKCS #11 PIN. If this property is set to the value `any`, then the getPin method in SSLCallback interface is invoked. If a PKCS #11 token login is required and neither the Login callback property nor the PIN property are set, the SSL session fails. This property can be set application-wide using the SSLServiceProvider context. This property cannot be retrieved once it has been set.
- **ORBuserData** Specifies user data (string datatype). This is an optional property. Client code can set user data during ORB initialization and access it using SSLSessionInfo::getProperty method in the SSL callback implementation. This may be useful as a mechanism to store ORB-level context information that is otherwise not available through the SSLSessionInfo interface.
- **ORBuseEntrustID** Specifies whether to use the Entrust ID or the Sybase PKCS #11 token for authentication. This is a Boolean (true or false) property. If this property is set to false, Sybase PKCS #11 token properties are valid and Entrust-specific properties are ignored. If this property is set to true, Entrust-specific properties are valid and Sybase PKCS #11 token properties are ignored.
- **ORBentrustPassword** Specifies the password for logging in to Entrust with the specified user profile. This property is a null-terminated string, which is optional when the Entrust single-login feature is available and required when this feature is not available. If the password is required but not set, the getPin method in CtsSecurity::SSLCallback is invoked to get the Entrust password. If there is no callback or if the callback does not return a password, the SSL session fails.

- **ORBentrustIniFile** Specifies the path name for the Entrust INI file that provides information on how to access Entrust. This is required when the useEntrustID property is set to true.
- **ORBentrustUserProfile** Specifies an Entrust user profile path name. This property is optional when the Entrust single-login feature is available and required when this feature is not available.

Creating a manager instance

Creating the manager instance for an SSL session is exactly like creating a manager instance for a non-SSL session, except that instead of specifying an IIOP port for the manager session in the string_to_object method, you specify the secure IIOP (specify `iiops`) port (the IIOPS default port number for mutual client-server authentication is `9002`). You must specify a port that supports the at least the level of security specified by the QOP setting.

Retrieving session security information

To retrieve security information about the session, narrow the component object reference, *typeObj*, to an SSL session, *sslSession*. Then use the getSessionInfo method to retrieve the session information from the SSL session and create an object reference for the session information. Use individual get methods to retrieve information about each SSL session property.

Note You cannot use the getName, getPassword, getAuthenticationStatus, getListener, getPeerAddress, getHostName. These methods are inherited from the SessionInfo interface.

```
// Obtain SSLSession information from
// typesObj.
CtsSecurity::SSLSession_var sslSession =
    CtsSecurity::SSLSession::
    _narrow(typesObj);
CtsSecurity::SSLSessionInfo_var
    sslSessionInfo = sslSession->getSessionInfo();
```

```
// Obtain user data (similar usage in user's
// SSL callback implementation)
String_var currentUserData =
    sslSessionInfo->getProperty("userData");

// Obtain details about server's certificate.
CtsSecurity::X509Certificate_var serverX509=
    sslSessionInfo->getPeerCertificate();
cout << "Connected to server: <<
    serverX509->getSubjectDN() << endl";

// get details about my certificate
CtsSecurity::X509Certificate_var clientX509=
    sslSessionInfo->getCertificate();
```

Creating an SSL callback component

An SSL callback component is a component that the client uses to execute callback methods. A callback method is a method that responds to SSL requests from EAServer. An SSL callback component resides on the client machine. To create an SSL callback, you must create a component DLL or shared library and deploy it on the client machine in a directory specified by the *PATH* environment variable. You can create the component in the same manner that you would create any other server-side component—using EAServer Manager and a C++ IDE.

You must specify the component DLL or shared library by using the setGlobalProperty method in the CtsSecurity::SSLServiceProvider interface to set the *callbackImpl* global property. For information, see “Set the quality of security services and global properties” on page 66.

Implementing callback methods

Although default implementations of the following callback methods are included with the EAServer client ORB, you can implement your own logic for these callback methods. To implement the default response for callback methods, code them to return the CORBA::NO_IMPLEMENT exception.

- **getCertificateLabel** The user is prompted with the available certificate labels and asked to choose one of them for client authentication.
- **getCredentialAttribute** The EAServer SSL client runtime engine retrieves credential attributes from the user on request.

- **getPin** The user is prompted with the PKCS #11 token or Entrust password information and asked to provide a PIN for logging into the PKCS #11 token or Entrust.
- **trustVerify** The user is prompted with server certificate information and asked to determine if the server certificate chain can be trusted and if the SSL session can proceed.

For more information about these callback methods, see the CtsSecurity::SSLCallback interface in the interface repository documentation. The interface repository documentation can be viewed in a Web browser by connecting to your server with this URL:

```
http://yourhost:yourport/ir/
```

where *yourhost* is the EAServer's host name and *yourport* is the HTTP port number.

Example

The *sample/ClientSSL* subdirectory in your EAServer installation contains an example program that installs an SSL callback to interact with the user.

CHAPTER 7

Using SSL in PowerBuilder Clients

You can create PowerBuilder clients that connect to EAServer using SSL connections, using techniques similar to that used in other client types. Since PowerBuilder connects to EAServer using the C++ client ORB, SSL in PowerBuilder requires a full EAServer C++ client installation.

You can configure the settings required for SSL connections using two techniques:

- **By setting ORB properties** When using this technique, the required SSL settings must be known in advance, and your application can connect only to servers that use certificates issued by a known, trusted certificate authority. To use this technique, set the required properties in the options string for the Connection or JaguarORB object. The SSL options for PowerBuilder clients are the same as listed for C++ clients in "ORB properties for secure sessions" on page 67.
- **By using the SSLServiceProvider interface** The SSLServiceProvider interface allows your application to determine what options are available at runtime. In addition, you can supply a callback class with methods that supply settings as needed and respond to exceptional cases. For example, the client ORB invokes callback methods if the application specifies an invalid certificate password or if a connection is made to a server that uses a certificate issued by an unknown certificate authority. PowerBuilder provides implementations of the SSLServiceProvider and SSLCallback objects that you can use in PowerScript®. You can create your own callback implementation by creating a standard custom class user object inherited from the SSLCallback object and implement the callback functions you need.

For more information on using these APIs, see the *Application Techniques* manual in the PowerBuilder documentation.

CHAPTER 8

Using SSL in ActiveX Clients

Beginning with version 3.5, EAServer allows you to use SSL connections between ActiveX clients and EAServer. Among other security features, SSL provides for certificate-based authentication of the server, optional certificate-based authentication of the client, and optional encryption of data transmitted over the network.

For more information about EAServer and SSL, see Chapter 13, “Managing Keys and Certificates.”

For more information on ActiveX clients, see Chapter 20, “Creating ActiveX Clients,” in the *EAServer Programmer’s Guide*.

Requirements

SSL support in ActiveX clients is provided through the C++ client ORB. Therefore, a complete C++ client installation is required, including the *%JAGUAR_CLIENT_ROOT%\DB* directory that contains the client side security database, SSL support libraries, and the stand-alone Security Manager.

The client installation must also be correctly configured to load the native EAServer client libraries when you run your application. See the *EAServer Installation Guide* for more information. Specifically, at runtime, the JAGUAR_CLIENT_ROOT variable must specify the full path to the EAServer client runtime installation directory, and the location of the EAServer client libraries must be in your system's shared library or DLL search path.

Warning! Make sure you select the C++ Runtime and SSL Runtime options when installing the EAServer client runtime. SSL support in ActiveX applications requires the files installed by these options.

Establishing a secure session

To create a secure session between your application and EAServer, you must configure SSL settings before using one of the standard techniques to instantiate proxies for EAServer components.

You can configure the settings required for SSL connections using two techniques:

- **By setting ORB properties** When using this technique, the required SSL settings must be known in advance, and your application can connect only to servers that use certificates issued by a known, trusted certificate authority.
- **By using the SSLServiceProvider interface** The SSLServiceProvider interface allows your application to determine what options are available at runtime. In addition, you can supply a callback class with methods that supply settings as needed and respond to exceptional cases. For example, the client ORB invokes callback methods if the application specifies an invalid certificate password or if a connection is made to a server that uses a certificate issued by an unknown certificate authority.

Applications that run without user interaction typically configure SSL settings with the ORB properties. Interactive applications typically use the SSLServiceProvider interface and install a callback. When a callback is installed, you can rely on user interaction in the callback methods to configure necessary settings. For example, if the certificate password has not been supplied, the ORB invokes the getPin callback method.

If you have correctly configured the required SSL settings, then you can use any of the standard techniques to instantiate proxies as described in "Instantiating proxies using CORBA-style interfaces" in Chapter 20, "Creating ActiveX Clients," of the *EAServer Programmer's Guide*. Proxies are created in a secure session as long as you connect to a listener that supports the requested level of security.

Using the SSLServiceProvider interface

The CtsSecurity.SSLServiceProvider interface provides setGlobalProperty and getGlobalProperty methods to set and retrieve the SSL properties listed in Table 8-1 on page 78. After initializing an ORB instance, you can instantiate a proxy for the SSLServiceProvider interface with the ORB.resolve_initial_references method, as shown below:

```
Dim orbRef As JaguarTypeLibrary.ORB
Dim ssp As CtsSecurity.SSLServiceProvider
Dim CORBAObj As Object ' Generic unnarrowed CORBA object

' Initialize the ORB
Set orbRef = New JaguarTypeLibrary.ORB
orbRef.Init ("")

' Get a proxy for the SSLServiceProvider
Set CORBAObj = _
    orbRef.resolve_initial_references("SSLServiceProvider")
Set ssp = CORBAObj.Narrow_("CtsSecurity/SSLServiceProvider")
```

You can then call the setGlobalProperty method to set properties, as in the example below:

```
ssp.setGlobalProperty("qop", "sybpks_intl");
```

Properties set with the SSLServiceProvider interface affect all ORB instances used by the application. However, if an equivalent property has been set when initializing an ORB instance, the ORB property value takes precedence.

You can retrieve property values using the getGlobalProperty method, which returns a JCollection instance. For example, this code retrieves the value of the availableQop property, which specifies the list of valid settings for the qop property:

```
Dim availQop As JaguarTypeLibrary.JCollection
```

```
' Retrieve available QOP settings and populate combo box
Set availQop = ssp.getGlobalProperty("availableQOP")

Dim iter As Integer

comboQOP.Text = "<none>"
Call comboQOP.AddItem("<none>", 0)

For iter = 1 To availQop.Count
  Call comboQOP.AddItem( _
    Format(availQop.Item(iter - 1)), iter)
Next iter
```

When retrieving properties that take a single value, the value is returned in a JCollection with one item.

SSL properties

Table 8-1 lists the ORB and SSLServiceProvider properties that govern the use of SSL. In addition, you need to connect to a server address that can support your chosen level of security, as described in “Secure server addresses” on page 81.

Some properties, if not set or set incorrectly, cause the ORB to invoke an SSL callback method. If you do not install an SSL callback, the default callback implementation aborts the connection attempt.

Table 8-1: SSL Properties

Property name for ORB.init	Property name for SSLServiceProvider	Description
-ORBpin	pin	Always required when using SSL. Specifies the PKCS #11 token PIN. This is required for logging in to a PKCS #11 token for client authentication and for retrieving trust information. This property cannot be retrieved. If not set, set to "any", or set incorrectly, the ORB invokes the getPin callback method.

Property name for ORB.init	Property name for SSLServiceProvider	Description
-ORBcertificateLabel	certificateLabel	Required when using mutual authentication. Specifies the client certificate to use if the connection requires mutual authentication. The label is a simple name that identifies an X.509 certificate/private key in a PKCS #11 token. If the property is not set and the connection requires mutual authentication, the ORB invokes the getCertificateLabel callback method, passing the list of available certificate names as an input parameter.
-ORBqop	qop	Always required when using SSL. Specifies the name of a security characteristic to use. See "Choosing a security characteristic" on page 80 for more information.
-ORBuserData	userData	Specifies user data (String datatype). This is an optional property. Client code can set user data during ORB initialization and access it using SSLSessionInfo::getProperty method in the SSL callback implementation. This may be useful as a mechanism to store ORB-level context information that is otherwise not available through the SSLSessionInfo interface.
-ORBuseEntrustID	useEntrustID	Specifies whether to use the Entrust ID or the Sybase PKCS #11 token for authentication. If this property is set to "false" (the default), Sybase PKCS #11 token properties are valid and Entrust-specific properties are ignored. If this property is set to "true", Entrust-specific properties are valid and Sybase PKCS #11 token properties are ignored.
-ORBentrustUserProfile	entrustUserProfile	Specifies the full path to the file containing an Entrust user profile. This property is optional when the Entrust single-login feature is available and required when this feature is not available. If not set, the ORB invokes the getCredentialAttribute callback method.
-ORBentrustPassword	entrustPassword	Specifies the password for logging in to Entrust with the specified user profile. This property is optional when the Entrust single-login feature is available and required when this feature is not available. If the password is required but not set or set incorrectly, the ORB invokes the getPin callback method. This property cannot be retrieved.

Property name for ORB.init	Property name for SSLServiceProvider	Description
-ORBentrustIniFile	entrustIniFile	Specifies the path name for the Entrust INI file that provides information on how to access Entrust. This is required when the useEntrustid property is set to true. If not set, the ORB invokes the getCredentialAttribute callback method.
-ORBAXSSLCBComponent	*none*	The PROGID for an ActiveX component that acts as an SSL callback.
none	callbackImpl	DLL, package, and component name of a C++ pseudocomponent that acts as an SSL callback, specified as: `myDLL/myPackage/myComponent` See "Implementing an SSL callback" on page 82 for more information.
none	availableQop	Retrieve only. A list of available security characteristics. The qop property can be set only to values that appear in this list.
none	availableQopDesc	Retrieve only. A list of descriptions for the available security characteristics, in the same order as listed in the value of the availableQop property.
none	entrustReady	Retrieve only. Returns "true" if Entrust PKI software is available on the client, "false" otherwise.
none	loginTimeout	The time in seconds before the login to the Sybase certificate database expires. The default timeout is indefinite. Before the login times out, the certificate password is cached and used for multiple SSL connections. In other words, the PIN must be presented only once before the timeout expires or the client program terminates, whichever occurs first.

Choosing a security characteristic

To use SSL, you must specify a value for the qop property in ORB properties or using the SSLServiceProvider interface. Set the qop to the name of an available security characteristic. The characteristic describes the CipherSuites the client uses when negotiating an SSL connection. When connecting, the client sends the list of CipherSuites that it uses to the server, and the server selects a cipher suite from that list. The server chooses the first cipher suite in the list that it can use. If the server cannot use any of the available CipherSuites, the connection fails.

Chapter 12, “Security Configuration Tasks” describes the security characteristics that are provided with EAServer. At runtime, you can retrieve a list of characteristics and their descriptions by retrieving the availableQop and availableQopDesc properties.

Set the qop to “sybpks_none” to prevent any use of SSL on a connection. This setting can be useful if you have set the qop globally for all ORBs using the SSLServiceProvider interface, and you want to override the qop for an individual ORB instance.

Secure server addresses

The client ORB connects only to a server listener that uses an equivalent or greater level of security as requested in the qop setting. When you use ORB.string_to_object to instantiate a SessionManager::Manager proxy, the listener specified by the server address must use a security profile that matches the client’s qop setting.

For more information on instantiating proxies, see “Instantiating proxies using CORBA-style interfaces” in Chapter 20, “Creating ActiveX Clients,” of the *EAServer Programmer’s Guide*.

Other useful ORB properties

The following ORB properties are not required in programs that use SSL, but affect the behavior of programs that use SSL:

Property	Description
-ORBLogFile	Specifies the name of a file to receive error logging information from the client ORB. There is no default, and logging is enabled by specifying a file name as this property. The file name is recreated each time a client program is run with the same setting. You can also specify a log file by setting the JAG_LOGFILE environment variable. (The latter is useful when troubleshooting an executable).
-ORBretryCount	The number of times to retry a connection attempt that has failed; the default is 5. You can also configure this property by setting the -ORB_RETRYCOUNT environment variable.

Implementing an SSL callback

When developing applications that interact with end users and support SSL, you should provide an SSL callback. The ORB invokes callback methods when required SSL settings have not been configured, or a setting has an incorrect value.

The callback can respond to exceptional conditions, such as server certificates that have expired. When using mutual authentication, the callback getCertificateLabel method allows you to present available certificates to the end user for them to choose. Lastly, the callback simplifies the handling of retry logic in the case where the user enters an invalid certificate password.

You can install a C++ callback or an ActiveX callback, but not both.

An ActiveX SSL callback must implement the methods in the CtsSecurity.SSLCallbackIntf interface. To install the callback, add a setting for the -ORBAXSSLCBComponent property in the ORB initialization string passed to the Orb.init method, as in the example below:

```
Dim orbOptions as String
orbOptions = "-ORBAXSSLCBComponent=mySSLCBProj.mySSLCBComponent,"
orbOptions = orbOptions & "-ORBqop=sybpks_intl"
Set orbRef = New JaguarTypeLibrary.ORB
orbRef.Init (orbOptions)
```

The SSLCallbackIntf methods are as follows:

- **getCertificateLabel** Called when the session requires mutual authentication and a certificate label has not been provided in ORB properties or in SSLServiceProvider global properties.

  The Visual Basic syntax of this method is:

  ```
  Public Function getCertificateLabel( _
     ByVal sessionInfo As Object, _
     ByVal labels As JaguarTypeLibrary.JCollection _
     ) As String
  ```

  where:

  - *sessionInfo* contains details of the potential SSL session. You can retrieve information about the session using the getProperty method.
  - *labels* is a JCollection containing available certificate labels. The callback must return one of them, or raise an ActiveX error to abort the connection attempt.

- **getCredentialAttribute** Called when additional information is required to use an Entrust certificate, such as the path to the Entrust profile file, or the path to the *entrust.ini* file.

  The Visual Basic syntax is:

  ```
  Public Function getCredentialAttribute( _
    ByVal sessionInfo As CtsSecurity.SSLSessionInfo, _
    ByVal attr As Long, _
    ByVal attrValues As JaguarTypeLibrary.JCollection _
    ) As String
  ```

  where:

  - *sessionInfo* contains details of the potential SSL session. You can retrieve information about the session using the getProperty method.
  - *attr* is one of the following numeric codes for the type of request:

Attr value	To request
CtsSecurity. CRED_ATTR_ENTRUST_INIFILE (1)	The full path and file name of the Entrust initialization file, which is usually *%SYSTEMROOT%\entrust.ini.*
CtsSecurity. CRED_ATTR_ENTRUST_USERPROFILE (2)	The full path and file name for the Entrust profile (*.epf* file).

  - *attrValues* is not currently used.

  getCredentialAttribute must return a String containing the requested information, or raise an ActiveX error to abort the SSL session.

- **getPin** Called when the certificate password has not been specified in ORB or SSLServiceProvider properties, or if the supplied password was incorrect.

  The Visual Basic syntax of this method is:

  ```
  Public Function getPin( _
    ByVal sessionInfo As Object, _
    ByVal timedOut As Boolean _
    ) As JaguarTypeLibrary.JCollection
  ```

  where:

  - *sessionInfo* contains details of the potential SSL session. You can retrieve information about the session using the getProperty method.

- *timedOut* value is true if a time limit was set for caching the certificate password and the time has expired (time limits are set as the loginTimeout property in the SSLServiceProvider interface).

The implementation should check the tokenName property of the SSLSessionInfo instance to determine whether the requested password is for the Sybase certificate database or for an Entrust profile, then clearly identify which password is required when prompting the user.

Your implementation can raise an ActiveX error to abort the connection attempt.

The getPin method must return the characters of the PIN as individual items in a JCollection instance. The following Visual Basic code shows how to populate a JCollection (*coll* in the example) with characters from a string (*pin* in the example):

```
Dim coll As JCollection
Set coll = New JCollection

Dim c As Byte
Dim iter As Integer
For iter = 1 To Len(pin)
  c = Asc(Mid(pin, iter, 1))
  coll.Item(iter - 1) = c
Next iter
```

- **trustVerify** Called when the correct PIN for the certificate database has not been set, or if the server has presented a questionable certificate. The callback response determines whether the connection is allowed and, optionally, whether the certificate should be added to the local EAServer client certificate database.

  The Visual Basic syntax of this method is:

  ```
  Public Function trustVerify( _
    ByVal sessionInfo As CtsSecurity.SSLSessionInfo, _
    ByVal reason As Long _
    ) As Long
  ```

  where:

  - *sessionInfo* contains details of the potential SSL session. You can retrieve information about the session using the getProperty method.
  - *reason* is a numeric code from Table 8-2:

Table 8-2: trustVerify reason codes

Reason code	Description
CtsSecurity. REASON_CHAIN_INCOMPLETE (1)	Server's certificate chain is incomplete. The ORB cannot complete the chain using the CA certificates in the Sybase certificate database.
CtsSecurity. REASON_UNKNOWN_CA (2)	The root CA in the server's certificate chain is not listed in the Sybase certificate database.
CtsSecurity. REASON_CHAIN_EXPIRED (3)	At least one certificate in the server's certificate chain has expired.
CtsSecurity. REASON_TRUSTDBPINNOTSET (4)	The password for the certificate database has not been set. Return CtsSecurity.TRUST_FAILED to cause the ORB to call the getPin callback method.
CtsSecurity. REASON_TRUSTDBLOGINFAILED (5)	The password for the certificate database was incorrect. Return CtsSecurity.TRUST_FAILED to cause the ORB to call the getPin callback method.

trustVerify must return one of the integer codes listed in Table 8-3:

Table 8-3: trustVerify return codes

Return code	Specified response
CtsSecurity.TRUST_ONCE (1)	Accept the certificate, but only trust for one SSL connection.
CtsSecurity.TRUST_FAIL (2)	Fail the session, or if the reason is REASON_TRUSTDBPINNOTSET (4) or REASON_TRUSTDBLOGINFAILED (5), call the getPin method.
CtsSecurity.TRUST_ALWAYS (3)	Accept the certificate and add the server's CA to the list of trusted CAs in the Sybase certificate database.
CtsSecurity.TRUST_NEVER (4)	Reject the connection and mark the CA as not trusted in the Sybase certificate database.
CtsSecurity.TRUST_SESSION (5)	Trust the server certificate chain only during this client program's sessions. If the client program is restarted, the certificate chain is not trusted.
CtsSecurity.TRUST_FAIL_SESSION (6)	Reject the certificate now and any time it reappears during the life of this client program. Do not mark the certificate as untrusted in the Sybase certificate database.

Your implementation of the getPin, getCertificateLabel, and getCredentialAttribute method should allow the user to cancel the connection attempt. In response to a user cancel, raise an ActiveX error exception to abort the SSL session. In Visual Basic, you can do this by raising an error with vbObjectError as the error number. If you provide an error description, and error logging has been enabled with the -ORBlogFile Orb property, the error description is written to the log. After an SSL session is cancelled, the client program receives a connection-fail error as it would from any other failed connection attempt.

For more information about these callback methods, see the documentation for the CtsSecurity::SSLCallback interface in the generated Interface Repository documentation.

Retrieving session security information

The CtsSecurity.SSLSession and CtsSecurity.SSLSessionInfo classes allow you to determine whether SSL is used on connections from a proxy to the server, and if so, retrieve the SSL session settings. The code below illustrates the sequence of calls:

```
  ... deleted code to set ORB ssl properties,
     create session, instantiate proxy myComp ...
  Dim sslSess As CtsSecurity.SSLSession
  Dim sslSessInfo As CtsSecurity.SSLSessionInfo
  sslSess =   myComp.Narrow_("CtsSecurity/SSLSession")
  On Error Go To noSSLError
  Set sslSessInfo = _
    sslSess.getSessionInfo.Narrow_( _
      "CtsSecurity/SSLSessionInfo")
noSSLError:
     ... an error raised by getSessionInfo most likely
         means that the proxy does not use SSL ...
```

You can narrow the proxy for any CORBA object to CtsSecurity/SSLSession to obtain information about the session in which the proxy was created. When narrowing the SSLSession proxy to CTSSecurity/SSLSessionInfo, the proxy server raises an error if the session is not using SSL.

The SSLSessionInfo methods allow you to determine the SSL session properties, such as the server's address, the client certificate in use, the server certificate in use, and so forth. For more information, see the generated Interface Repository documentation for the CtsSecurity::SSLSessionInfo interface.

Example: inspecting SSL session properties

The Visual Basic fragment below prints a description of the SSL session in which a SesssionManager::Session proxy was created:

```
Public Function SessionDetails( _
  title As String, _
  obj As JaguarTypeLibary.Object _
  )
   Me.Caption = title
   Call clearOutput
   output (title & ":" & vbCrLf)
   Dim sslSess As CtsSecurity.SSLSession
   Dim sslSessInfo As CtsSecurity.SSLSessionInfo
```

```
   Dim host, port, prop As String
   Dim inError As Boolean
   inError = False

   On Error GoTo errorGetSession
   Set sslSess = obj.Narrow_("CtsSecurity/SSLSession")
   Set sslSessInfo = sslSess. _
     getSessionInfo.Narrow_("CtsSecurity/SSLSessionInfo")

   On Error GoTo errorGetProperties
   host = sslSessInfo.getProperty("host")
   port = sslSessInfo.getProperty("port")

   output ("Connected to " & host & ":" & port & vbCrLf)

   prop = sslSessInfo.getProperty("cipherSuite")
   output ("Negotiated CipherSuite: " & prop & vbCrLf)

   ' Print the server certificate details
   On Error GoTo errorGetServerCert
   Dim cert As CtsSecurity.X509Certificate
   Set cert = sslSessInfo.getPeerCertificate().Narrow_("CtsSecurity/X509Certi
ficate")
   output (vbCrLf & "Server certificate info:" & vbCrLf)
   output (certInfo(cert))

   ' Print the client certificate details
   On Error GoTo errorGetClientCert
   Set cert = sslSessInfo.
      getCertificate().Narrow_("CtsSecurity/X509Certificate")


   output (vbCrLf & "Personal certificate info:" & vbCrLf)

   output (certInfo(cert))
   inError = True ' Fall through error cases
' Error handling code. Labels are in reverse order of the
' On Error activations.
' Code to handle errors when retrieving the client certificate.
' Sessions will not have a client certificate unless mutual
' authentication is used. So, this is not necessarily an error.
errorGetClientCert:
   If Not inError Then
     inError = True
     output (vbCrLf & "No personal certificate in use." & vbCrLf)
   End If
' Code to handle errors raised when getting the server certificate.
```

```
' If a connection uses SSL, it should at least have a server certificate,
' so errors raised are likely due to coding errors.
errorGetServerCert:
   If Not inError Then
     inError = True
      output (vbCrLf & "** Error retrieving server certificate properties. **
" _
         & vbCrLf)
   End If
' Code for errors raised when retrieving session properties. Any error
' raised is likely due to a coding error.
errorGetProperties:
   If Not inError Then
      inError = True
      output ("Error retrieving SSL session properties." & vbCrLf)
   End If
' Code for errors raised when retrieving the session information.
' Errors here most likely mean that the connection does not use SSL.
errorGetSession:
   If Not inError Then
      inError = True
      output ("SSL not used on this connection.")
   End If

   ' All error handlers must fall through to here.
   Me.Show

End Function
```

Example: inspecting X.509 certificate properties

The previous example calls the following function to print a description of an SSL certificate represented in a CtsSecurity::X509Certificate instance:

```
Private Function certInfo( _
  cert As CtsSecurity.X509Certificate _
  ) As String

   Dim description As String
   Dim prop As String

   description = ""

   prop = cert.getSubjectDN()
   description = description _
```

```
      & "  Subject name: " & prop & vbCrLf

   prop = cert.getIssuerDN()
   description = description _
     & "  Issuer name: " & prop & vbCrLf

   description = description _
     & "  Not valid before: " & Format(cert.getNotBefore()) & vbCrLf

   description = description _
     & "  Not valid after: " & Format(cert.getNotAfter()) & vbCrLf

   certInfo = description

End Function
```

CHAPTER 9

Creating and Using Custom Security Components

This chapter describes how to use custom components to perform security tasks such as user authentication and authorization. These features allow you to create your own components to customize EAServer security and to integrate with third-party enterprise security software such as Netegrity SiteMinder.

Using a custom authentication service

You can install your own component to authenticate clients for any EAServer. For example, if you require the client user name to match a remote database user name, you can code the component to retrieve the client user name and password and attempt to log in to the remote database.

The component must implement the CtsSecurity::AuthService IDL interface, and you must set the com.sybase.jaguar.server.authservice server property to specify the name of your component (this property must be set using the Advanced tab in the Server Properties dialog).

This interface contains the method, checkSession. Your code for this method can check the client's user name and password and the status of other authentication checks, that is, whether the client's credentials have passed OS authentication or SSL authentication checks. Your code can perform additional authentication checks and auditing. For more information, see the documentation for the CtsSecurity::AuthService IDL interface.

A sample Java implementation is provided in the EAServer *html/classes/Sample/AuthServiceDemo* directory in your EAServer installation. A sample C++ implementation is available in the *sample/AuthServiceDemo* subdirectory.

Maintaining authenticated sessions

EAServer provides a mechanism by which applications can extend and maintain the authenticated session beyond the lifetime enforced by EAServer. This mechanism uses the methods CtsSecurity::SessionInfo::setName and CtsSecurity::AuthService::getCallerPrincipal.

If these methods are implemented, then you must also handle the authorization of the user by either implementing a role service or authorization service. The internal role checking performed by EAServer will not work unless the alternate user name is added to the authorized user's list for the role. As the alternate user name that is set using the setName API can be dynamic, the role service or authorization service should work in tandem with the authentication service to authorize the user.

CtsSecurity::SessionInfo::setName is a method that can be called only when your custom authentication component is running. When this method is called from the custom authentication component, the server sets the reference to the authenticated security credentials. When the client needs to be authenticated again, the custom authentication component returns the original principal name by calling CtsSecurity::AuthService::getCallerPrincipal(string alternate_name).

The CtsSecurity::SessionInfo::setName method has no effect if clients obtain component instances using CSIv2. If you are using CSIv2, you must use a JAAS module in addition to an authentication or other component. See Chapter 10, "Using the JAAS API" for more information.

For more information, see the documentation for the CtsSecurity::AuthService and CtsSecurity::SessionInfo IDL interface.

Retrieving HTTP session information

In a custom authentication component implemented in Java, you can call the com.sybase.jaguar.server.Jaguar.getHttpServletRequest() method to retrieve the HTTP servlet request (if any) that triggered the authentication event. This method returns null if the authentication event is not associated with an HTTP request (for example, if the authentication is for a component invocation).

Using a custom role service

You can install your own component that performs access control based on role membership. The component must implement the CtsSecurity::RoleService IDL interface. Your custom role service evaluates user membership in EAServer roles, so authorization in your application is still dependent on the role names associated with a package, component, method, or Web resource collection. Using a role service eliminates the need to define role memberships in EAServer Manager. For example, you might code your component to retrieve role membership information from a database.

Creating a role service

The role service must be a stateless component that implements the CtsSecurity::RoleService IDL interface:

```
interface RoleService {
     boolean isMember(
          in CtsSecurity::SessionInfo sessionInfo,
          in string role);
 };
```

isMember checks if the authenticated client is a member of the role. The client's credentials are obtained from sessionInfo. The server first checks if the role is defined in the repository. If the role is defined, then membership checks are performed. If the role is not defined, the server assumes that the user is not a member of the role, and the role service is invoked. The result from this method is cached by the server, where it can be referenced for the same client/role combination, provided the internal cache has the relevant information.

For more information, see the generated CtsSecurity::RoleService IDL interface documentation.

Installing the role service

Use EAServer Manager to enable the role service. You can write an implementation of the RoleService interface and configure a server-level role service by setting the `com.sybase.jaguar.server.roleservice` property to the URL that accesses the component that implements this interface. Set this property using the Advanced tab of the Server Properties window.

There are two accepted forms for the URL:

- You can set the URL to the *EAServerPackage/EAServerComponent* if the component is a Java CORBA, C++ CORBA, stateless COM or PowerBuilder NVO. The component must be installed in the server.

  For example, to set the role check service, set the server-level property to `com.sybase.jaguar.server.roleservice=`*Security/RoleService* where *Security* is the name of the package that contains the *RoleService* component that implements the RoleService interface.

- You can access Java CORBA and C++ CORBA components using the pseudocomponent object URL. The syntax for a Java pseudocomponent is:

```
pseudo://java/JavaClass/EAServerPackage/EAServerComponent
```

The syntax for a C++ pseudocomponent is:

```
pseudo://cpp/SharedLibraryName/EAServerPackage/EAServerComponent
```

You can also set the authorization service property to the pseudocomponent object URL. For example, you can set the server-level authorization service to:

```
pseudo://cpp/libAuthorize/Security/RoleService
```

where *libAuthorize* is the name of the shared library that contains the C++ Security/RoleService component's implementation.

Components implemented for pseudocomponent access must be thread-safe. Pseudocomponents cannot be refreshed. You must restart the server to refresh the role service component.

For more information on pseudocomponents, refer to Chapter 34, "Creating and Using EAServer Pseudocomponents," in the *EAServer Programmer's Guide*.

Using a custom authorization service

You can create and install your own component to authorize clients to access resources (packages, Web applications, or applications) on any EAServer.

Deciding whether to use the authorization services and role service

Using an authorization service offers greater control than using a role service, but the API is more complicated than the role service API.

The role service acts server-wide, and evaluates user membership in declared EAServer roles associated with a resource (package, component, method, or Web resource collection).

An authorization service can control access to all resources on a server, or only those in a particular application, Web application, or package. With the authorization service, you can allow or deny access to resources with no dependencies on roles configured in EAServer.

You can use both a role service and an authorization service. For example, you may wish to use a role service to preserve the ability to configure role-based resource permissions in EAServer Manager, but use the authentication service to create audit logs of user access to resources.

Creating the authorization service

An authorization service component must implement the CtsSecurity::AuthorizationService IDL interface, and be stateless to support refresh. It must be one of:

- Java CORBA
- PowerBuilder Non-Visual User Object (NVO)
- C++ CORBA
- Component Object Model (COM)

Usage

```
interface AuthorizationService {
     boolean isAuthorized(
             in CtsSecurity::SessionInfo sessionInfo,
             in StringSeq resource,
             in StringSeq roles,
```

```
                                in boolean isMember,
                                 in long permTimeDelta);
```

isAuthorized checks if the client is authorized to access a resource. The client's credentials can be obtained from sessionInfo.

resource is the entity the client is trying to access. The resource is represented as an ordered array of strings, and each string represents a scoped entity. A string starts with one of these prefixes:

- A: – application
- WA: – Web application
- P: – package
- C: – component
- M: – method
- S: – servlet
- HM: – HTTP method (GET, PUT, POST, and so on)

For example, if the resource being accessed is a servlet or a JSP that belongs to a Web application, which belongs to an application, then the array might contain the following string sequence:

```
A:ApplicationName; WA:WebApplicationName; S:servletName; HM:httpMethod;
```

roles lists all the roles associated with the resource (if any). The server first checks if the role is defined in the repository. If the role is defined, then membership checks are performed and if the user is in at least one of the roles, the authorization check succeeds. isAuthorized is still invoked, and the caller can audit the resource access. isMember is set to AUTH_OK to indicate that the authorization succeeded. If a role is not defined, it is assumed that the user is not a member of the role.

If the user is not a member of all the roles, then isMember is set to AUTH_FAILED. isAuthorized then determines whether to authorize the client. isAuthorized returns true if the user is allowed access to the resource, and returns false otherwise.

permTimeDelta is the time difference in seconds, since the last time isAuthorized was invoked for this particular user and resource combination. This value can be used by the authorization component logic to determine whether to audit the event. A value of zero (0) implies that the isMember was not determined from the internal permission cache. A positive value indicates that the isMember was determined from the internal permission cache. permTimeDelta is always less than or equal to the server-wide authorization permission cache timeout value (see the `com.sybase.jaguar.server.authorization.permcachetimeout` property).

For more information, see the generated documentation for the CtsSecurity::AuthorizationService IDL interface.

Installing the authorization service

Use EAServer Manager to install the authorization service component in the server, application, package, or Web application. There are two ways in which you can make the authorization service available to all components on EAServer:

- Allow multiple packages, Web applications, or applications to share the same authorization service by setting the same value for the authorization service component. If the application utilizes a particular authorization service, then all components accessed by the application also utilize the same authorization service. To configure an authorization service:
  - At the package level, set the com.sybase.jaguar.package.authorization.service property to the URL for the component that implements this interface in the Advanced tab of the Package Properties window.
  - At the Web application level, set the com.sybase.jaguar.webapplication.authorization.service property to the URL for the component that implements this interface in the Advanced tab of the Web Applications Properties window.
  - At the application level, set the com.sybase.jaguar.application.authorization.service property to the URL for the component that implements this interface in the Advanced tab of the Applications Properties window.

- Enable the interface on the entire server. Set the com.sybase.jaguar.server.authorization.service property to the URL for the component that implements this interface in the Advanced tab of the Server Properties window. Packages, Web applications, and applications can utilize the authorization service.

Component URLs for the authorization service

There are two accepted forms of the URL:

- For all component types, the URL can be set to the *EAServerPackage/EAServerComponent*; the component must be installed in the server.

  For example, to set the authorization service at the server level, set the server-level property to:

  `com.sybase.jaguar.server.authorization.service=`*Security/Authorizer*

  Where *Security* is the name of the Jaguar package that contains an EAServer component called *Authorizer* that implements this interface.

- Java CORBA and C++ CORBA components can be accessed using the pseudocomponent object URL. The syntax for a Java pseudocomponent is:

```
pseudo://java/JavaClass/EAServerPackage/EAServerComponent
```

  The syntax for a C++ pseudocomponent is:

```
pseudo://cpp/SharedLibraryName/EAServerPackage/EAServerComponent
```

  You can also set the authorization service property to the pseudocomponent object URL. For example, set the server-level authorization service to:

```
pseudo://cpp/libAuthorizer/Security/Authorizer
```

  where *libAuthorizer* is the name of the shared library that contains the C++ *Security/Authorizer* component's implementation.

  Components implemented for pseudocomponent access must be thread-safe, and you must restart EAServer to refresh the component.

  For more information on pseudocomponents, see Chapter 34, "Creating and Using EAServer Pseudocomponents," in the *EAServer Programmer's Guide*.

Supporting external single sign-on providers

EAServer allows integration with external single sign-on authentication software such as Netegrity SiteMinder. EAServer includes custom security components to support Netegrity, and you can implement support for other services by implementing your own custom security components.

The API CtsSecurity::CallerPrincipalService allows you to implement a component that tells EAServer the effective user ID when authentication occurs outside of EAServer. For details on creating and installing a caller principal service, see the HTML reference documentation for this interface in file *html/ir/CtsSecurity.html* in your EAServer installation directory.

The com.sybase.jaguar.server.http.sso server property specifies whether sign-on occurs externally. Set this property to true if you are using an external single-sign on provider.

Netegrity SiteMinder Integration

EAServer supports integration with Netegrity SiteMinder security software. Netegrity SiteMinder provides single sign-on and centralized management of Web, database, and software resources in enterprise applications. For more information, see the Netegrity Web site at http://www.netegrity.com/.

The following configurations are supported:

- Web access to EAServer through a secure reverse-proxy server. This configuration provides global single sign-on support for all applications and servers that are protected by the proxy server, as well as centralized user and user rights management. In this configuration, no direct user connections are allowed to EAServer. Instead, users access EAServer via the proxy server. Users log in to the secure proxy server using basic (user name plus password) authentication or by presenting an SSL certificate. This configuration requires a reverse-proxy server that supports Netegrity single sign-on, such as Apache with the Netegrity Web Agent installed or the Netegrity Secure Proxy Server.

- Direct client access to EAServer with Netegrity authentication. In this configuration, users present their login credentials (user name and password or SSL certificate) to EAServer. The Netegrity agent installed in EAServer forwards the credentials to the Netegrity Policy Server for validation. While this configuration does not support global single sign-on, it does allow you to take advantage of centralized user and user-rights management provided by the Netegrity Policy Server.
- Mixed access, which is a combination of these two approaches. For example, you can enable access through a proxy server to provide global single sign-on support to Web client users, while still supporting direct IIOP or IIOPS connections to EAServer from other client applications.

EAServer integration with SiteMinder is provided by Java Authentication and Authorization Service (JAAS) modules installed in EAServer, along with custom role service and caller principal service components. These components use the Netegrity Agent API to connect to the Netegrity Policy Server to verify user credentials, login status, and role membership.

When using Netegrity, EAServer authorization is based on the EAServer roles that are associated with components and Web resources, with role membership evaluated by the Netegrity Policy Server. The required roles for resource access are determined based on the component or Web application properties, as set in EAServer Manager or jagtool. When a resource requires role membership for access, EAServer calls the Netegrity role service, which determines whether the user is a member of the required role based on settings maintained in the Netegrity Policy Server.

These JAAS login modules are provided for Netegrity/EAServer integration:

- An HTTP login module, which allows EAServer Web applications to support Netegrity single sign-on in reverse-proxy configurations.
- A X.509 certificate login module, which validates client SSL certificates presented to EAServer by forwarding them to the Netegrity Policy Server.
- A basic login module, which validates client user names and passwords presented to EAServer by forwarding them to the Netegrity Policy Server.

Configuring your security scenario

The Netegrity integration login modules are defined in the Netegrity JAAS configuration file, *netegrity_jaas.cfg*. On Windows, this file is installed in the *ini* subdirectory of your EAServer installation. On UNIX platforms, the file is installed in the *config* directory. You must configure the security scenario that you want to use by modifying the attributes of each login module in this file. The attributes for each scenario are listed below:

- For Web only access via the Netegrity Secure Proxy Server, use these settings in *netegrity_jaas.cfg*:

Login module	Attribute
HTTP LoginModule	Requisite
X.509 LoginModule	Optional
Basic LoginModule	Optional

- For direct client access using basic authentication without support for single sign-on, use these settings in *netegrity_jaas.cfg*:

Login module	Attribute
HTTP LoginModule	Optional
X.509 LoginModule	Optional
Basic LoginModule	Requisite

- For mixed access, use these settings in *netegrity_jaas.cfg*:

Login module	Attribute
HTTP LoginModule	Sufficient
X.509 LoginModule	Sufficient
Basic LoginModule	Sufficient

Configuring the SiteMinder Policy Server

The following configuration can be performed in the Netegrity Policy Server User Interface Console. For detailed instructions, see the Netegrity documentation. These settings are required for all scenarios.

❖ **Policy Server setup**

1 Create a Web agent named easagent, configured with the Policy Server host name and password.

2 Create a user directory with all the user names to be authenticated. Also, add a user “Anonymous” with password “Anonymous”. The anonymous user is required to allow IIOP login without user credentials, such as for a client accessing a message-driven bean.

3 If you use client certificates in your application, enter the common name of each certificate in the user directory.

4 Configure an authentication scheme to match your Netegrity configuration scenario, as described in “Authentication methods for EAServer and SiteMinder” on page 107.

5 Configure a domain named Sybase that uses the user directory. Create a realm named “EAS” with these properties:

   a Agent is “easagent”.

   b Resource Filter is “/EAS”.

   c Default Resource Protection is “Unprotected”.

   d Authentication Scheme matches the scheme you configured previously.

6 For the EAS realm, create a rule named “DummyResource” with resource “/DummyResource”. This rule must be enabled with the “Allow Access” option selected. This rule is the default resource for authentication.

7 For the EAS realm, create additional rules for each EAServer role with the following properties:

   a Set the resource to:

      `/ROLE/`*`role-name`*

      Where *role-name* is the EAServer role name, as displayed in EAServer Manager. For example, “Admin Role” in EAServer requires the resource `/Role/Admin Role`.

   b Set Web Agent Actions to “Get, Post, Put.”

   c Enable the rule and select the “Allow Access” option.

8 Create a new policy, for example, Policy01. For each role used in your application, create mappings for the client user names and certificate common names that belong to the role. These mappings are used for role-based authorization of resource access.

9 If you use client certificates in your application, configure the certificate mapping properties. Create a mapping for each issuing certificate, that is, the distinguished name of each root certificate that corresponds to a certificate authority used by your application. Map this distinguished name to the user directory type that matches the user directory that you created earlier. For each mapping, select the Single Attribute mapping option, and select the Common Name (CN) as the attribute to map.

10 To ensure the changes you have made take effect, flush the Policy Server cache.

Configuring reverse-proxy access to EAServer

To support Netegrity single sign-on in your application, you must configure a compatible reverse-proxy server. EAServer has been tested with the Apache Web server running as a reverse-proxy with the Netegrity Web Agent installed. See the Netegrity SiteMinder *Web Agent Installation Guide* and *Web Agent Guide* for instructions on configuring Apache to run with the Netegrity Web Agent installed.

Reverse-proxy access requires the additional Policy Server settings described below.

❖ **Policy Server configuration for reverse-proxy server access**

Use the Netegrity Policy Server User Interface Console to perform this configuration. For detailed instructions on each step, see the Netegrity documentation:

1 Create a new Web agent to represent the proxy server, for example, ApacheAgent.

2 Create an Agent Conf object for the proxy server agent. Highlight the ApacheDefaultSettings object, then create a new object from it. Set the DefaultAgentName parameter to match the name of the Web agent created in step 1, for example, “ApacheAgent.”

3 Create a Host Conf object for the proxy server. Highlight the DefaultHostSettings object, then create a new object from it. Configure the Policy Server IP address and listener ports to match your installation.

4 Configure authentication schemes to match your Netegrity configuration scenario. For user name/password access, configure a scheme that uses BASIC or FORM authentication. For client certificate authentication, configure a scheme that uses X.509 template authentication. For FORM and X.509 schemes, configure the proxy server itself as the Server name setting.

5 Create a new realm for the Web agent that represents the proxy server with these settings:

   a For Agent, select the name of the Web agent, for example, “ApacheAgent”.

   b Add “/” to the resource filter.

   c For Default Resource Protection, select Unprotected.

   d Select an appropriate authentication scheme.

6 Create a rule named “All” in the realm with these settings:

   a Set the resource to “*”.

   b Select “Get, Post, Put” for the Web agent actions.

   c Select Allow Access.

   d Select Enabled.

7 In the policy configuration, set up mappings for the All rule to include the client user names and certificate common names that are used in your application.

8 To ensure the changes you have made take effect, flush the Policy Server cache.

Enabling Policy Server logging

To troubleshoot problems, enable debug logging in the Policy Service Management Console. Select “Log to File” and “Append” options. Do not select “Log to Console”.

Configuring EAServer for SiteMinder security

❖ **Configuring EAServer to use SiteMinder security**

1 Install the Netegrity JAAS configuration file into your server. The file is *netegrity_jaas.cfg*, located in the EAServer *ini* subdirectory on Windows platforms and *config* subdirectory on UNIX platforms. Install the JAAS module as follows:

   a Using EAServer Manager, display the Server Properties dialog box. On the Security tab, set the JAAS Configuration File to the full path to the *netegrity_jaas.cfg* file.

   b If you are running a server other than the preconfigured Jaguar server, display the Advanced tab. Set the com.sybase.jaguar.server.jaas.section property to `Jaguar`. If this property is not present, add it.

2 Follow the instructions for your platform below to copy necessary files from the Netegrity SDK installation to the JDK installation that you use to run EAServer.

   On UNIX platforms, verify the JDK location by checking the values of the JAGUAR_JDK13 or JAGUAR_JDK14 variables in the EAServer *bin/setenv.sh* file. Copy these files from the Netegrity SDK installation to the JDK *jre/lib/sparc* subdirectory:

   - *libsmagentapi.so*
   - *libsmjavaagentapi.so*

   On Windows platforms, verify the JDK location by checking the values of the JAGUAR_JDK13 or JAGUAR_JDK14 variables in the EAServer *bin\setenv.bat* file. Copy these files from the Netegrity SDK installation to the JDK *jre\bin* subdirectory:

   - *smAgentAPI.dll*
   - *smJavaagentapi.dll*

3 Copy the following JAR files from the Netegrity SDK to the *java/lib* subdirectory of your EAServer installation:

   - *smjavaagentapi.jar*
   - *smjavaskd2.jar*

4 On the Advanced tab in the Server Properties dialog box, set the property com.sybase.jaguar.server.callerprincipalservice to:

```
pseudo://java/com.sybase.jaguar.security.netegrity/CtsSecurity/NetegrityCallerPrincipal
```

5 On the Advanced tab in the Server Properties dialog box, set the property com.sybase.jaguar.server.roleservice to:

```
pseudo://java/com.sybase.jaguar.security.netegrity/CtsSecurity/NetegrityRoleService
```

6 Also on the Advanced tab, set the properties listed in the table below:

Property	Value
com.sybase.jaguar.server.http.sso	If you have configured single sign-on support using a reverse-proxy server, set to `true` to enable external single sign-on support in EAServer. If your configuration allows direct client connections to EAServer, set to `false`.
com.sybase.jaguar.server.smAgentName	The agent name used in the SiteMinder Policy Server, for example, “easagent”.
com.sybase.jaguar.server.smAgentPassword.e	The agent password used to connect to the SiteMinder Policy Server. The password is stored in encrypted form in the EAServer repository.
com.sybase.jaguar.server.smServerAddress	The host name of the SiteMinder Policy Server.
com.sybase.jaguar.server.smAgentDebug (optional)	Optionally set to `true` to enable debug message logging from the Netegrity integration components installed in EAServer.
com.sybase.jaguar.server.smAuthorizationPort (optional)	Authorization port for the SiteMinder Policy Server. If not set, the default is 44443.
com.sybase.jaguar.server.smAuthenticationPort (optional)	Authentication port for the SiteMinder Policy Server. If not set, the default is 44442.
com.sybase.jaguar.server.smAccountingPort (optional)	Accounting port for the SiteMinder Policy Server. If not set, the default is 44441.
com.sybase.jaguar.server.server.smTimeout (optional)	The SiteMinder cache lifetime limitation in seconds. If not set, the default is two times of EAServer Authorization cache timeout, specified by the server property com.sybase.jaguar.server.authorization.permcachetimeout
com.sybase.jaguar.server.smSize (optional)	The SiteMinder cache size. If not set, the default is 600.

7 For each EAServer Web application, display the Web Application Properties in EAServer Manager. Configure the authentication method as described in “Authentication methods for EAServer and SiteMinder” on page 107.

Authentication methods for EAServer and SiteMinder

You must configure the Netegrity and EAServer authentication methods differently depending on whether you allow direct log in to EAServer. If you allow direct login to EAServer, configure the EAServer and SiteMinder authentication methods to match according to Table 9-1. If you use FORM authentication, the login and error page must be set and deployed in EAServer. Do not mix certificate based authentication with user name/password based authentication. In other words, all EAServer Web applications must use FORM or BASIC, or all must use CLIENT-CERT.

Table 9-1: Authentication methods for scenarios that allow direct EAServer login

EAServer authentication method	SiteMinder authentication scheme type
FORM	BASIC
BASIC	BASIC
CLIENT-CERT	X.509

If you use a reverse-proxy server to support Netegrity single sign-on, use BASIC in EAServer. In SiteMinder, use BASIC, FORM, or X.509 as required by the application. In this case, authentication is performed within the reverse-proxy server and the Netegrity setting supersedes the EAServer setting.

CHAPTER 10

Using the JAAS API

Introduction

The Java Authentication and Authorization Service (JAAS) provides a framework and standard programming interface for authenticating users and assigning privileges. JAAS is based on the Pluggable Authentication Module (PAM) standard, which extends the access-control architecture of the Java 2 platform to support user-based authentication and authorization.

JAAS support is provided in EAServer as an alternative authentication mechanism. EAServer supports user name-password based JAAS authentication. The code-level authorization component of JAAS is not supported in this version of EAServer.

JAAS required when using corbaname URLs in clients If an EJB client uses corbaname or corbaloc interoperable naming URLs, you must specify the user name and password using the JAAS API. See Chapter 9, "EAServer EJB Interoperability," in the *EAServer Programmer's Guide* for more information on corbaname URLs and other interoperability features.

See the Java software Web site at http://www.javasoft.com/products/jaas for more information about JAAS.

There are several new terms that are used throughout this chapter:

Principal represents a user identity that is used to gain access to a computing service. Typically, a user's login name or public key.

Credentials represents a security attribute of a principal. Typically, a user's password or public-key certificate. The credential is set in the subject when a principal is authenticated successfully.

Subject is an entity that has one or more principals and corresponding security attributes.

A *login context* is a JAAS framework for developing applications independent of underlying authentication technology.

A *login module* is an authentication module that can be plugged in under a Java application using JAAS framework. The module implements the JAAS javax.security.auth.spi.LoginModule interface. It performs any authentication either on its own or by interacting with any external authentication service such as Kerberos.

A *callback* is a mechanism by which a login module retrieves authentication parameter values needed for authentication from the Java application. The callback is implemented in a Java application to pass required parameters to the login module. It implements the javax.security.auth.callback interface.

The *JAAS configuration file* specifies:

- One or more authentication modules for an application
- The order in which authentication modules are invoked
- Other parameters and options

This is the interaction between an application, login module, and the JAAS configuration file:

1 The Java application program instantiates a login context that consults the JAAS configuration file to load all of the login modules configured for that application.

2 The login module requests the Java program to provide the user name and password using the JAAS callback mechanism.

3 The login module executes custom code to authenticate the user and set up the subject with valid principals and credentials if successfully authenticated.

The subject can then be used to gain access to controlled resources or to perform privileged actions.

Requirements

If your client application uses JAAS APIs, you need JDK 1.3 or later. If your client is JAAS-enabled, you do not need JDK 1.3 or later on EAServer unless you are using JAAS in EAServer as well.

JAAS in EAServer

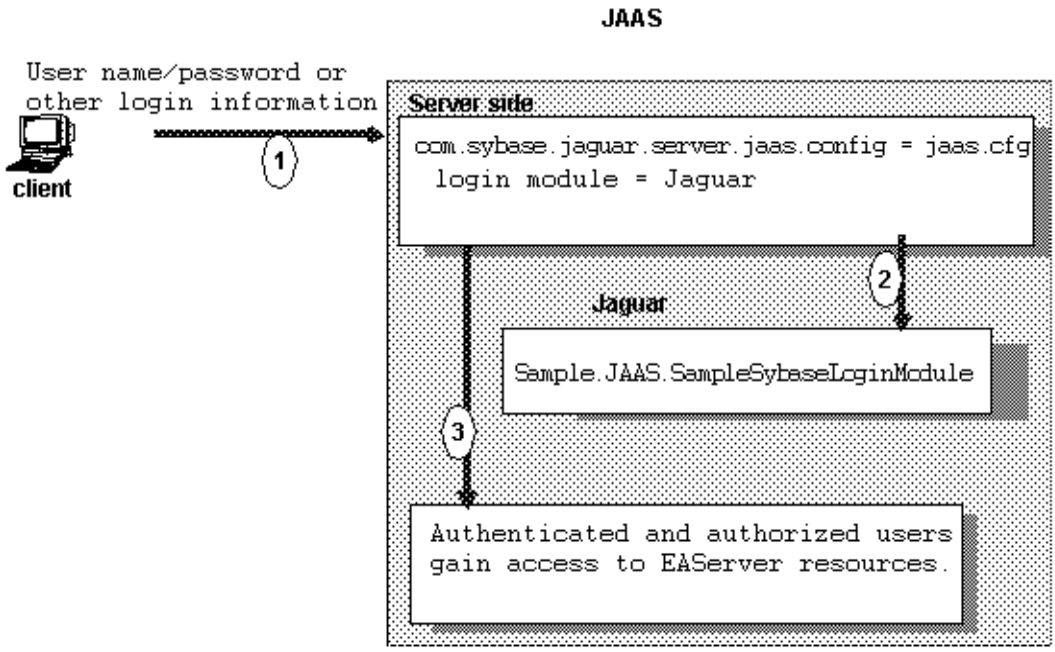
Over time, you may need to modify or replace authentication infrastructure due to deficiencies, enhancements, or applications requiring a different security policy. EAServer support for JAAS login modules simplifies replacement and modification of the underlying authentication mechanism.

Configure server-wide login modules that are used to authenticate clients trying to gain access to applications, Web applications, and servlets/JSPs. Figure 10-1 illustrates how JAAS is enabled on EAServer. The com.sybase.jaguar.server.jaas.config server property (defined in EAServer Manager) points to the JAAS configuration file, which determines the login module to use for a specific server. The configuration file requires a section specified by the server property com.sybase.jaguar.server.jaas.section. If you do not set this property, the section name must match the server name.

Based on the contents of the configuration file, EAServer invokes any specified login modules. If a login module is not defined, then JAAS is bypassed and the server uses the regular mechanism, if any, for authentication. For example, if credentials are passed to a server and no login module is defined, the server uses operating system authentication, if enabled.

If a login module is defined, it overrides any other authentication service that may be installed, and passes the request for authentication to the login module.

Figure 10-1: EAServer login design



Enabling JAAS for a server

To enable JAAS for a server, you must specify the JAAS configuration file and section name in the server properties. EAServer uses the login module in that section for authentication.

❖ **Enabling JAAS for a server:**

1 Highlight the Servers folder.

2 Highlight the server for which you are identifying the configuration file.

3 Select File | Properties, and highlight the Security tab.

4 In the JAAS Configuration File window, enter the name of the JAAS configuration file, or use the browse button to search for the file.

You can verify the JAAS configuration file setting in the Advanced tab by viewing the com.sybase.jaguar.server.jaas.config property.

Note To disable JAAS, remove the entry from the JAAS Configuration File window.

5 Optionally specify the name of a section in the JAAS configuration file by setting the property com.sybase.jaguar.server.jaas.section on the Advanced tab. If you do not specify a section name, the file must contain a section with the same name as the server.

This message indicates that JAAS is disabled, or there is a JAAS error; for example, the configuration file is not valid, or there is a problem loading the login module:

```
May 30 16:30:35 2001: Note: No configuration found for 'Jaguar' in the JAAS
configuration file.
May 30 16:30:35 2001: WARNING: JAAS setup for Authentication is ignored.
```

Your EAServer installation contains a sample JAAS configuration file, *jaas.cfg*, in the *html/classes/Sample/JAAS* directory.

Retrieving additional user session details in a JAAS login module

EAServer allows you to retrieve additional IIOP or HTTP user session information when using the JAAS API to install custom security implementations. A sample that demonstrates these features is installed in the *html/classes/Sample/JAAS* directory of your EAServer installation.

When authenticating HTTP client access, you can retrieve the servlet request details associated with the client request, returned as an instance of javax.servlet.http.HttpServletRequest . EAServer provides a JAAS callback implementation in class com.sybase.jaguar.security.HttpServletRequestCallback. To retrieve servlet sessions, add an instance of this class to the callback stack in your implementation's login method. Call the HttpServletRequestCallback.getHttpServletRequest() method to retrieve the servlet request. The method returns null if the request is not an HTTP request.

When authenticating IIOP client access, you can retrieve details about the client session as an instance of the CtsSecurity/SessionInfo built-in component. To do so, add the EAServer callback class com.sybase.jaguar.security.SessionInfoCallback to the callback stack in your implementation's login method. Call the SessionInfoCallback.getSessionInfo() method to retrieve the CtsSecurity.SessionInfo class instance that describes the user session. For details on the CtsSecurity.SessionInfo methods, see the documentation in the following file in your EAServer installation:

```
html/ir/CtsSecurity__SessionInfo.html
```

JAAS on the client

EAServer includes a JAAS login module com.sybase.jaguar.security.auth.module.JaguarLoginModule. It uses the JAAS callback mechanism to obtain the client's user name and password and generate credentials.The credentials are passed to the server when the client attempts to invoke any component on the server. This login module must be used if you want your EJB client or Java CORBA client to obtain credentials from the user using JAAS.

❖ **To enable JAAS on the client:**

1 Make sure a login module is defined in the JAAS configuration file that requires com.sybase.jaguar.security.auth.module.JaguarLoginModule, for example:

```
/*
This section can be used by Jaguar clients which invoke
JaguarLoginModule to setup proper credentials.
*/
JaguarClient{
  com.sybase.jaguar.security.auth.module.JaguarLoginModule required
  debug=true;
};
```

2 Set the name of the JAAS configuration file in the Java interpreter's `-Djava.security.auth.login.config` property on the client's machine.

3 Create an instance of the login module, using code like this. In this fragment, JaguarClient is the name of the the name of the JAAS configuration section that requires the module com.sybase.jaguar.security.auth.module.JaguarLoginModule:

```
LoginContext lc = null;
try {
lc = new LoginContext("JaguarClient", new MyCallbackHandler());
} catch (LoginException le) {
System.err.println("Cannot create LoginContext. "
+ le.getMessage());
System.exit(-1);
} catch (SecurityException se) {
System.err.println("Cannot create LoginContext. " + se.getMessage());
System.exit(-1);
}
Initial Context ic = new InitialContext();
... regular code to look and invoke methods on an EJB.
```

JAAS for connectors

The J2EE connector architecture enables you to write portable Java applications that can access multiple transactional enterprise information systems. A resource adapter is a specialized connection factory that provides connections for: EJBs, Java servlets, JSPs, and CORBA-Java components.

Each resource adapter has a set of managed connection factories with their own property files. The Java Connection Manager (JCM) classes create the connection factories and manage a pool of connections for a resource adapter. You can use JAAS to authenticate a resource adapter, resource principal, or the application component's caller principal when accessing enterprise information systems.

See "Configuring connectors" in the *EAServer System Administration Guide* for more information about connectors.

Container-managed authentication

EAServer is responsible for connector security when a component accessing enterprise information systems requests container-managed authentication. In this case, "container-managed" means "EAServer-managed."

To enable container-managed authentication from EAServer Manager:

1 Highlight the Component folder.

2 Highlight the component for which you are establishing container-managed authentication.

3 Select File | properties.

4 Select the Advanced tab.

5 Set the deployment property of the component to container managed:

```
res-auth=Container
```

Container is the default setting.

If `res-auth=Application,` a null subject is passed to the connector and connection security is handled by the application.

Basic password authentication

EAServer 4.0 supports basic password authentication. For connectors, this means establishing a user name and password for the resource adapter's managed connection factory (MCF). To set basic password authentication from the Advanced tab, click com.sybase.jaguar.connector.auth-mechanism and set auth-mech-type to "basic-password."

To establish a user name and password for the resource adapter's MCF from EAServer Manager:

1 Select the Connectors folder.

2 Select the connector for which you are setting a user name and password authentication.

3 Select the MCF for which you are enabling authentication.

4 Select File | Managed Connection Factory Properties.

5 Select the Security Properties tab and enter the user name and password for the MCF.

If you do not supply a user name and password in the resource adapter's MCF, they are obtained from the CallerPrincipal (the user name and password that are retrieved from the calling component).

Enabling JAAS-based authentication for connectors

To use JAAS for authentication, you must enable JAAS on the EAServer where the resource adapter is located. See "JAAS in EAServer" on page 111.

An entry in the login configuration file is identified by the name of the resource adapter for which JAAS is used. If container-managed authentication is set, any component that tries to obtain a connection from resource adapter's connection factory is authenticated by the login module defined by the configuration file entry.

Samples and debugging

A JAAS sample that includes a login module, configuration file, and so on, ships with EAServer. See the *README* file in the *html/classes/Sample/JAAS* directory of your EAServer installation.

You can enable the login module's internal tracing by setting "debug=true" in the JAAS configuration file.

For additional debugging information for JAAS-based custom authentication in EAServer, set com.sybase.jaguar.server.jaas.debug=true in the server's Advanced tab.

For additional debugging information for connectors, set com.sybase.jaguar.server.jcm.trace=true in the server's Advanced tab.

CHAPTER 11

Deploying Applications Around Proxies and Firewalls

Proxy servers are typically used to isolate computers on a corporate network from the Internet. Connections to and from the Internet must go through the organization's proxy servers. EAServer does not include proxy server software, but supports client connections through proxy servers to EAServer.

This chapter describes security features for clients that connect over the Internet or through proxy servers. You should understand the basics of your client model before reading this chapter.

Connecting through proxy servers

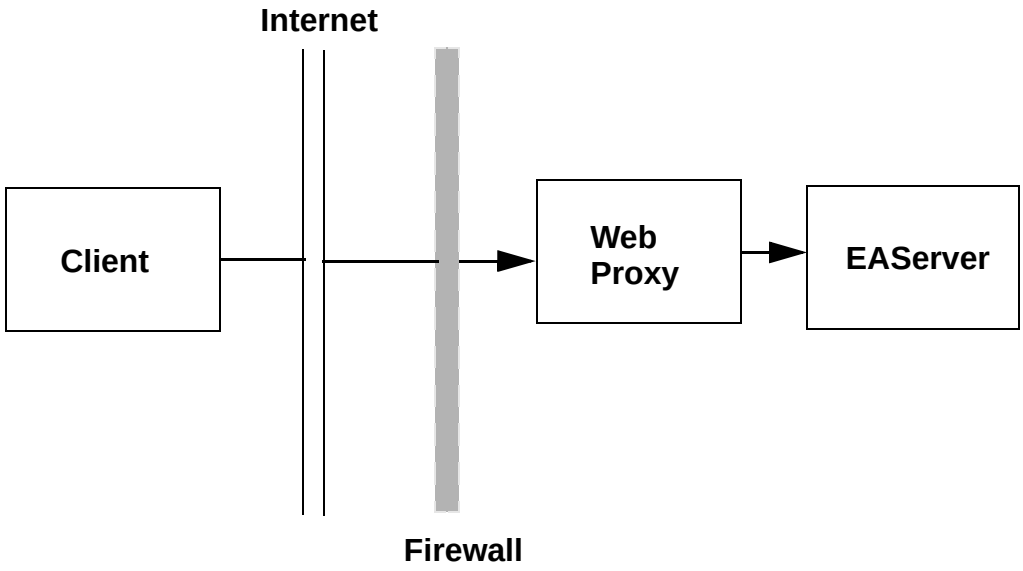
Proxy servers are typically used to constrain and secure connections from an organization's computers to sites that require connecting across the Internet. To enhance security, some network configurations require all Internet connections to go through a proxy server, including IIOP connections to EAServer.

EAServer supports two types of proxy servers for clients, Web proxies and reverse proxies.

Using Web proxies

Web proxies typically act as a gateway for outgoing connections from a group of workstations. Web proxies can be used to enhance network security, for example, a proxy may constrain which servers clients can connect to and which protocols may be used, and log outgoing connections. Web proxies may also be used to improve network performance, by caching the results of frequently executed Web requests. Web proxies are also referred to as HTTP-connect-based proxies. Figure 11-1 illustrates how clients connect to servers through a Web proxy:

Figure 11-1: Connecting through a Web proxy



Clients connect to EAServer through a Web proxy as follows:

1 Using the HTTP protocol The client connects over the Internet to the Web proxy, embedding the destination server address inside a specially formatted HTTP connect request.

2 The Web proxy connects to the host and port indicated in the initial HTTP connect request.

3 Subsequent traffic is forwarded unchanged between the client and server until the connection is closed.

Java applets can use the built-in proxy configuration provided by Web browsers such as Netscape Navigator. See your Web browser's documentation for information on configuring proxy addresses. For applets running in a Web browser, HTTP and HTTPS-tunnelled IIOP connections automatically use the browser's proxy connection settings. HTTP-tunnelled IIOP connections go through the browser's configured HTTP proxy. HTTPS-tunnelled IIOP connections go through the browser's configured secure proxy.

Other applications must specify the Web proxy address by setting the Web proxy host and port in the properties described below.

Properties that affect Web proxy use

Table 11-1 describes the client properties that configure connections that must be opened through a Web proxy. You must set these properties in addition to any properties that you would set to connect directly to EAServer.

Table 11-1: Properties that affect Web proxy use

C++/ActiveX/ PowerBuilder property	CORBA property	EJB property	Specifies
`ORBWebProxyHost` or environment variable `JAG_WEBPROXYHOST`	`com.sybase.CORBA.WebProxyHost`	`com.sybase.ejb.WebProxyHost`	Specifies the host name or IP address of the Web proxy server. Does not apply to Java applets running in a Web browser, which use the proxy address specified by the browser's proxy configuration. There is no default for this property, and you must specify both the host name and port number properties.
`ORBWebProxyPort` or environment variable `JAG_WEBPROXYPORT`	`com.sybase.CORBA.WebProxyPort`	`com.sybase.ejb.WebProxyPort`	Specifies the port number at which the Web proxy server accepts connections. Does not apply to Java applets running in a Web browser, which use the proxy address specified by the browser's proxy configuration. There is no default for this property, and you must specify both the host name and port properties.

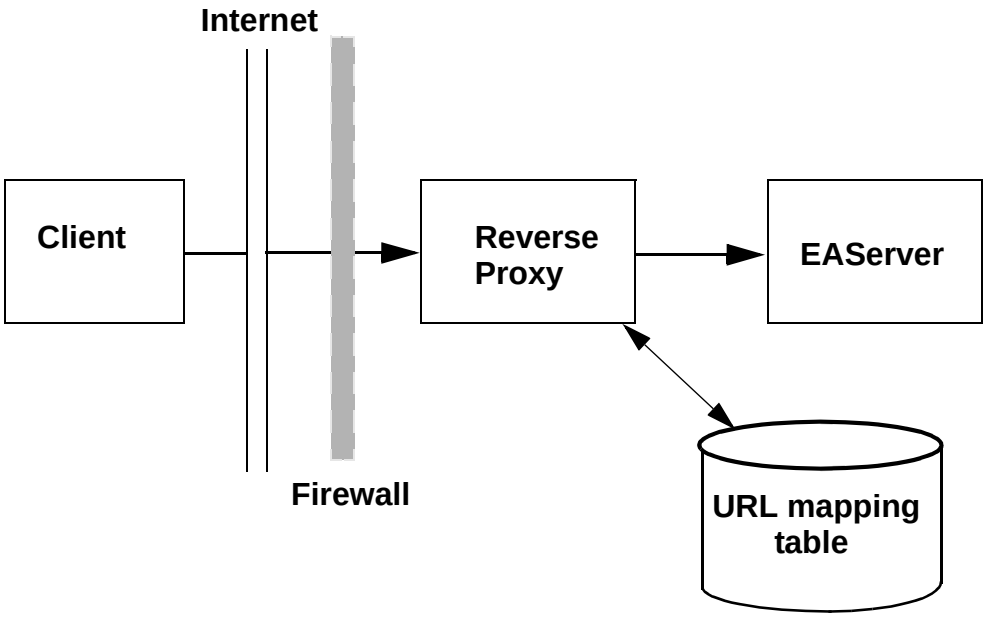
C++/ActiveX/ PowerBuilder property	CORBA property	EJB property	Specifies
`ORBHttp` or environment variable `JAG_HTTP`	`com.sybase.CORBA.http`	`com.sybase.ejb.http`	By default, the client ORB attempts to open IIOP connections, then attempts an HTTP-tunnelled connection if plain IIOP fails. Since Web proxy connections require HTTP tunnelling, set this to true to eliminate the performance overhead of trying plain IIOP connections before trying HTTP-tunnelled IIOP.
`ORBHttpExtraHeader` or environment variable `JAG_HTTPEXTRAHEADER`	`com.sybase.CORBA.HttpExtraHeader`	`com.sybase.ejb.HttpExtraHeader`	An optional setting to specify what extra information is appended to the header of each HTTP packet sent to the Web proxy server. There is no need to set this property unless your HTTP proxy server has special protocol requirements. By default, the following line is appended to each packet: `User-agent: Jaguar/`*`major.minor`* where *major* and *minor* are the major and minor version numbers of your EAServer client software, respectively. You can set this property to specify text to be included at the end of each HTTP header. If multiple lines are included in the setting, they must be separated by carriage return and line feed characters. If the setting does not include a "User-agent: " line, then the default setting above is included in the HTTP header.

C++/ActiveX/ PowerBuilder property	CORBA property	EJB property	Specifies
N/A.	`com.sybase.CORBA.useJSSE`	`com.sybase.ejb.useJSSE`	Use the Java Secure Sockets Extension (JSSE) classes for secure HTTP tunnelled (HTTPS protocol) connections. JSSE provides an alternative to the built-in SSL implementations when secure connections are needed from an applet running in a Web browser. Additional configuration may be required to use this option. See "Using Java Secure Socket Extension classes" on page 58 for more information.

Using reverse proxies

Reverse proxies typically act as a gateway for incoming connections to an organization's network servers, preventing direct connections from clients outside the firewall to servers inside the firewall. The reverse proxy can enhance security, by restricting protocols and logging connection activity. Reverse proxies may also act as caches to respond to common requests. In some cases, multiple reverse proxies may be deployed to cache results from one server, as a form of load balancing. Figure 11-2 shows how clients connect through a reverse proxy.

Figure 11-2: Connecting through a reverse proxy



Clients connect to EAServer through a reverse proxy as follows:

1 The client connects to the reverse proxy, and sends each IIOP packet tunnelled inside an HTTP or HTTPS packet. The destination server address is encoded in the HTTP packet header as:

```
GET /host/port/HIOP/1.0/...
```

Where *host* is the target EAServer host name, and *port* is the target EAServer port number.

2 The reverse proxy uses its URL mapping configuration (shown as a database in the figure) to determine the destination server address.

3 The reverse proxy opens a connection to the destination server, or reuses an existing connection, and forwards the request to the server, then forwards the response to the client.

Reverse-proxy configuration

For use with EAServer, you must configure your reverse proxy server's URL mapping table to recognize the EAServer addresses embedded in the HTTP requests sent by the client runtime. For each EAServer that clients can connect to through the server, configure a mapping for the following URL prefix:

```
GET /host/port/HIOP/1.0/
```

Where *host* is the target EAServer listener host name, and *port* is the target EAServer listener port number. For each EAServer that you deploy behind the reverse proxy, add a mapping for each IIOP, IIOPS, and Message Service listener address. If you deploy an EAServer cluster behind a reverse proxy, add mappings for each server in the cluster.

Properties that affect reverse proxy use

To connect through a reverse-proxy server, you can set the properties in Table 11-2. You must set these properties in addition to any properties that you would set to connect directly to EAServer.

Table 11-2: Properties that affect reverse proxy use

C++/ActiveX/ PowerBuilder property	CORBA property	EJB property	To indicate
`ORBProxyHost` or environment variable `JAG_PROXYHOST`	`com.sybase.CORBA.ProxyHost`	`com.sybase.ejb.ProxyHost`	Specifies the machine name or the IP address of the reverse-proxy server.
`ORBProxyPort` or environment variable `JAG_PROXYPORT`	`com.sybase.CORBA.ProxyPort`	`com.sybase.ejb.ProxyPort`	Specifies the port number of the reverse-proxy server, typically 80 for HTTP-tunnelled connections or 443 for SSL (HTTPS-tunnelled) connections.
`ORBHttp` or environment variable `JAG_HTTP`	`com.sybase.CORBA.http`	`com.sybase.ejb.http`	Set this property to true if the reverse-proxy server requires HTTP-tunneled connections. If you do not set this property, connections still go through, but only after the client ORB first tries to open an IIOP connection. Setting the property eliminates the overhead that is incurred by trying plain IIOP each time a connection is made.

C++/ActiveX/ PowerBuilder property	CORBA property	EJB property	To indicate
ORBforceSSL or environment variable JAG_FORCESSL	com.sybase.CORBA.forceSSL	com.sybase.ejb.forceSSL	Set this property to true if the connection to the reverse proxy must use SSL (HTTPS) tunnelling, but the connection from the proxy to the EAServer does not use SSL tunnelling.
ORBqop or environment variable JAG_QOP	com.sybase.CORBA.qop	com.sybase.ejb.qop	In applications that connect to a proxy using SSL (HTTPS) tunnelling, set the Quality Of Protection (QOP) to a security characteristic that matches the one supported by the reverse-proxy server. See “Configuring security profiles” on page 139 for more information. If the connection to the proxy server requires SSL, but the connection from the proxy does not, do not set the QOP; instead, set the forceSSL property to true. Do not set QOP in Java applets that use SSL. Instead, code the applet to connect to a listener that supports the required security level. See “Using SSL in Java applets” on page 41 for more information.

C++/ActiveX/ PowerBuilder property	CORBA property	EJB property	To indicate
N/A.	`com.sybase.CORBA.autoProxy`	`com.sybase.ejb.autoProxy`	In Java applets, set this property to true to enable connections to a reverse-proxy server. You must also configure your applet to download through the reverse-proxy server itself. The default is false. This property is ignored if the client is not a Java applet, or has not initialized the Java ORB with the ORB.init method that takes an Applet parameter. When automatic proxy is enabled, the ORB uses the applet's download address as the reverse-proxy server address. If the port number is 443, SSL (HTTPS tunnelling) is used; otherwise, HTTP tunnelling is used.

CHAPTER 12

Security Configuration Tasks

This chapter describes the most common declarative security mechanisms provided by EAServer and configured with EAServer Manager.

Configuring EAServer roles

EAServer's authorization model is based on *roles*, which are defined in EAServer Manager. Each role can include and exclude specific user names or digital IDs. If you use native operating system authentication, you can also include and exclude operating system group names; all users in the specified group are affected.

Roles are attached to EAServer packages and components. A package or component's role controls access as follows:

- If any roles are assigned to a package, the user must have all of these roles to use any component in the package.
- If any roles are assigned to a component, the user must have all of these roles to use the component.
- If roles are assigned to both a component and the package that contains it, the user must have all of the roles that are assigned to the package and component.

You must either refresh or restart EAServer for any role changes to take effect.

❖ **Refreshing EAServer**

1 Highlight the Roles folder.

2 Select File | Refresh.

❖ **Defining a new role**

1 Highlight the Roles folder.

2 Select File | New Role. Enter the required information in the subsequent dialogs:

- New Role – the name of the role you are defining.
- Description – the description, up to 255 characters, of the role.
- Owner – the owner of the role.

❖ **Deleting an existing role**

1 Highlight the Roles folder. You see a list of existing roles.

2 Highlight the role you want to delete.

3 Right-click the role and select Delete. This option is available only to the owner of the role or the jagadmin user.

4 Click Yes to confirm deletion of the selected role.

Note Only the owner or a member of the role named Admin Role can delete a role, except for Admin Role itself, which cannot be deleted. See "Admin role in EAServer" on page 135 for more information.

❖ **Modifying an existing role**

1 Highlight the Roles folder. You see a list of existing roles.

2 Highlight the role you want to modify.

3 Select File | Properties.

4 Make your modifications and click OK.

❖ **Adding an existing role, or creating and adding a new role to a package, component, or method**

1 Double-click the icon for the package, component, or method to expand the folders beneath it. Highlight the Role Membership folder.

2 Select File | Install Role. Then select one of the following options from the Role wizard:

- Install an Existing Role – a list of uninstalled roles appears in the dialog. Highlight the role to be installed and click OK.
- Create and Install a New Role – enter the name of the new role to be installed. Complete the role property sheet. The properties are described in "Defining a new role" on page 130.

Note A package, component, or method with no roles or role memberships defined has no access restrictions.

Assigning users and groups to roles

Each role can include and exclude specific user names and digital IDs. If you use native operation system authentication, you can also include and exclude operating system group names; all users in the specified group are affected.

❖ **Assigning authorized users to a role of a component or a package**

1 Double-click the component or package to which the role belongs.

2 Highlight the Roles folder.

3 Double-click the role to which you want to add authorized users.

4 Highlight the Authorized User folder.

5 Select File | Add Authorized User.

6 Enter the name of the authorized user in the dialog, and click Add Authorized User. On Windows, you can provide the name of the domain as part of the authorized user name; for example, *domain_name**user_name*. The user is authenticated using the domain name controller for that domain.

The user's name appears on the right side of the window when you highlight the Authorized Users folder.

To remove an existing authorized user, highlight the member and select File | Remove Member.

❖ **Assigning authorized groups to a role of a component or a package**

1 Double-click the component or package to which the role belongs.

2 Highlight the Roles folder.

3 Double-click the role to which you want to add authorized groups.

4 Highlight the Authorized Group folder.

5 Select File | Add Authorized Group.

6 Enter the name of the authorized group in the dialog, and click Add Authorized Group.

The group's name appears on the right side of the window when you highlight the Authorized Groups folder.

To remove an existing authorized group, highlight the member and select File | Remove Member.

Note The users and groups of a role are mapped to operating system users and groups. To validate users and groups, you must click Enable User and Group Validation from the server's Security property sheet. You can only add validated groups to roles. When Enable User and Group Validation is disabled, package and component authorizations stop at the user level. There is no attempt to check group level authorization.

❖ **Assigning authorized digital IDs (certificates) to a component or a package**

1 Double-click the component or package to which the role belongs.

2 Highlight the Roles folder.

3 Double-click the role to which you want to add authorized digital IDs.

4 Highlight the Authorized Digital IDs folder.

5 Select File | Add Authorized Digital ID.

6 A list of digital IDs appears. Double-click the name of the digital ID that you want to authorize, and click Add Authorized Digital ID.

Only certificates that appear in the EAServer Manager | Certificate folder | User Certificates folder and Other Certificates folder can be authorized. This requires that you install the certificate using EAServer Manager | Certificate folder. See Chapter 13, "Managing Keys and Certificates" for more information.

The user's name appears on the right side of the window when the Authorized Digital IDs folder is highlighted.

To remove an existing authorized digital ID, highlight the member and select File | Remove Member.

You can verify, export, or view information about an authorized digital ID by highlighting the digital ID and selecting the corresponding option from the file menu. See Chapter 13, “Managing Keys and Certificates” for more information about these options.

❖ **Excluding users from a component or a package**

1 Double-click the component or package to which the role belongs.

2 Highlight the Roles folder.

3 Double-click the role from which you want to exclude users.

4 Highlight the Excluded User folder.

5 Select File | Add Excluded User.

6 Enter the name of the excluded user in the dialog, and click Add Excluded User. On Windows, you can provide the name of the domain as part of the excluded user name; for example, *\\domain_name\user_name*. The user is authenticated using the domain name controller for that domain.

The user’s name appears on the right side of the window when the Excluded Users folder is highlighted.

To remove an existing excluded user, highlight the member and select File | Remove Member.

❖ **Excluding groups from a component or a package**

1 Double-click the component or package to which the role belongs.

2 Highlight the Roles folder.

3 Double-click the role from which you want to exclude groups.

4 Highlight the Excluded Group folder.

5 Select File | Add Excluded Group.

6 Enter the name of the excluded group in the dialog box, and click Add Excluded Group.

The group’s name appears on the right side of the window when you highlight the Excluded Groups folder.

To remove an existing excluded group, highlight the member and select File | Remove Member.

❖ **Excluding digital IDs (certificates) from a component or a package**

1 Double-click the component or package to which the role belongs.

2 Highlight the Roles folder.

3 Double-click the role from which you want to exclude digital IDs.

4 Highlight the Excluded Digital IDs folder.

5 Select File | Add Excluded Digital ID.

6 A list of digital IDs appears. Double-click the name of the digital ID that you want to exclude, and click Add Excluded Digital ID.

   Only certificates that appear in the EAServer Manager | Certificate folder | User Certificates folder and Other Certificates folder can be excluded. This requires you to install the certificate using EAServer Manager | Certificate folder. See Chapter 13, “Managing Keys and Certificates” for more information.

The user’s name appears on the right side of the window when the Excluded Digital IDs folder is highlighted.

To remove an existing excluded authorized digital ID, highlight the member and select File | Remove Member.

You can verify, export, or view information about an excluded digital ID by highlighting the digital ID and selecting the corresponding option from the file menu.

Determining authorization

The following order is used to determine role based authorization:

1 If the user is authorized, the search terminates and authorization is granted.

2 If the user is excluded, the user is declined access to the resource.

3 If the user is in an authorized group:

   a Check if the role is a member of the authorized group.

   b If this check succeeds, check if the role is a member of an excluded group list—if not, grant access to the resource.

Purpose of excluded lists

Excluded lists simplify the task of granting authorization to a small number of users by denying access to the users based on their user names and not the authorized groups to which they belong when using group-based authorization.

Note If a user is a member of an excluded user or group list, EAServer does not invoke the Role Service (CtsSecurity/RoleService) if defined for that server.

Predefined roles

EAServer includes a number of predefined, read-only roles that you can use to facilitate authorization to EAServer resources. Role names are case sensitive and include:

ServiceControl Prevents clients from invoking service components.

anonymous Allows access to an 'anonymous' user.

everybody Allows access to all authenticated users.

system Prevents access by any client. The system user is a member, so components with this role can run as EAServer services.

nobody Prevents all access to a method or component. No user is a member of this role, not even the EAServer system user.

Admin role in EAServer

Every EAServer contains an Admin package and an Admin role. You must be a member of the Admin role to run EAServer Manager.

Initially, only jagadmin is a member of this role. The jagadmin user can set up additional members.

Even though other users can belong to the Admin role and run EAServer Manager, only the jagadmin user can:

1 Set the following options from EAServer Manager | Servers folder | *server_name* | Properties | Security tab:

- The jagadmin password
- Enable OS Authentication
- Enable User & Groups Validation

2 Modify users, groups, or digital IDs belonging to the EAServer Manager | Roles | Admin role.

❖ **Granting permissions to EAServer roles**

Beginning with EAServer 5.0, members of the Admin role can use EAServer Manager or jagtool to grant permissions to other EAServer roles; for example, permission to start or shut down a server.

Note Although users with the Admin role can grant permission to other roles to perform certain tasks, these tasks must be performed using jagtool because only members of the Admin role can access EAServer Manager.

---

1 In EAServer Manager, expand the Roles folder, highlight the role to which you want to grant permissions, right-click, and select Properties.

2 In the Role Properties dialog box, select any of the tabs described below.

Application Authorities To grant users with the current role permission to create, modify, or delete an application, select Add Application, and enter the application name.

To remove an application from the list of those that users with this role have permission to access, highlight the application name, and select Delete Application.

Package Authorities To grant users with the current role permission to create, modify, or delete a package, select Add Package, and enter the package name.

To remove a package from the list of those that users with this role have permission to access, highlight the package name, and select Delete Package.

Server Authorities To authorize users with the current role permission to perform an action on the server, select the action:

- Restart Server
- Refresh Server
- Shut down Server

To revoke permission to perform an action, unselect the action.

Servlet Authorities To grant users with the current role permission to create, modify, or delete a servlet, select Add Servlet, and enter the servlet name.

To remove a servlet from the list of those that users with this role have permission to access, highlight the servlet name, and select Delete Servlet.

Web Application Authorities To grant users with the current role permission to create, modify, or delete a Web application, select Add Web Application, and enter the Web application name.

To remove a Web application from the list of those that users with this role have permission to access, highlight the Web application name, and select Delete Web Application.

For information about using jagtool to grant and revoke permissions, see the reference pages for the commands grantroleauth or removeroleauth in Chapter 12, "Using jagtool and jagant," in the *EAServer System Administration Guide*.

Configuring OS authentication

To enable OS authentication on EAServer:

1 From EAServer Manager, double-click the server you want to configure.

2 Select File | Properties.

3 Select the Security tab.

Enable OS Authentication – if selected, this option maps EAServer client users to operating system user names and passwords. You must supply a user name and password that is valid for the machine where the EAServer is running. For example, for UNIX, you would use network information service (NIS) passwords, and for Windows, you would use your Windows domain password. Windows users can provide a domain name as part of their user name; for example, *domain_name**username*.

For Windows, Active Directory Services style accounts are supported. The format of Active Directory Service (ADS) accounts is *username@domain*. For example, username@sybase.com.

To use OS authentication on Windows, the user who runs EAServer must be included in the "Administrators" group on your Windows machine.

❖ **Enabling OS authentication on UNIX**

- Select the Enable OS Authentication option on the Security tab.

❖ **Enabling OS authentication on Windows 2000**

1. Select Start | Settings | Control Panel.
2. Double-click Administrative Tools.
3. Double-click Local Security Settings.
4. In the left pane, click Local Policies.
5. Select and open User Rights Assignment.
6. Double-click Act as Part of the Operating System.
7. Click Add in the new pop-up window to add the desired users. This provides the required privileges to EAServer to authenticate a user by querying the underlying operating system.
8. Log out, then log back in to your Windows 2000 system to enable authentication.
9. From EAServer Manager, select Enable OS Authentication on the Server Properties Security tab.

❖ **Enabling OS authentication on Windows XP**

1. Select Start | Settings | Control Panel.
2. Double-click Administrative Tools.
3. Double-click Local Security Policy.
4. Expand the Local Policies folder, then select User Rights Assignment.
5. Double-click Act as Part of the Operating System.
6. In the new dialog box, click Add User or Group to add users.
7. In the Select Users or Groups dialog box:
   a. Click Object Types, and select Users.
   b. Click Locations, and select the network domain.
   c. Enter the user names.

   This provides the required privileges to EAServer to authenticate a user by querying the underlying operating system.
8. Log out, then log back in to your Windows XP system to enable authentication.
9. From EAServer Manager, select Enable OS Authentication on the Server Properties Security tab.

Note The password for the jagadmin account is always defined in EAServer Manager. Even if jagadmin is defined as an OS user name and OS authentication is enabled, the password defined in EAServer Manager is required to log in as jagadmin.

Configuring OS user and group authorization

Enable User & Groups Validation – if enabled this option, the user and group names are validated against their operating system user and group name before being added to any of the following folders located in the Role folder:

- Authorized User
- Authorized Group
- Excluded User
- Excluded Group

To enable user and group validation, select the Enable User and Groups Validation option on the server's Security tab.

Configuring security profiles

Security profiles define the security characteristics of a client-EAServer session. You assign a security profile to a listener, which is a port that accepts client connection requests of various protocols. EAServer can support multiple listeners. Clients that support the same characteristics can communicate to EAServer via the port defined in the listener.

Each security profile has an associated security characteristic. A security characteristic is a name that has a set of cipher suites associated with it. A security characteristic, along with the cipher suites, defines these characteristics of a client/server connection:

- **Protocol** All profiles use SSL version 3 as the underlying protocol. IIOPS and HTTPS traffic is tunneled through SSL.

- **Authentication** Whether or not authentication is used. Profiles can support:
  - No authentication – neither client nor server need to provide a certificate for authentication.
  - Server authentication – only the server needs to provide a certificate to be accepted or rejected by the client.
  - Client and server authentication – both the client and server supply certificates to be accepted or rejected by the other.
- **Encryption strength and method** Whether or not data is encrypted, and if so, the key strength and method of the encryption.
- **International use** All cipher suites are available domestically, but not all are suitable for export outside of the United States and Canada.
- **Hashing method** The method used to create the message digest.

For example, the cipher suite SSL_RSA_WITH_NULL_MD5 can be interpreted as:

SSL – the protocol used. All profiles use SSL.

RSA – the key exchange algorithm used.

NULL – no encryption.

MD5 – the hash method used to compute the message digest.

Table 12-1 and Table 12-2 clarify the relationship between cipher suite terminology and security characteristics.

Table 12-1: Cipher suite terms

Name	Defines	Description
SSL	Protocol	SSL protocol uses public-key encryption to establish secure Internet communications.
RSA DH_anon	Key exchange algorithm	RSA and DH (Diffie-Hellman) are public-key cryptography systems, which define both authentication and encryption: • RSA provides full encryption and authentication support. • DH_anon provides only encryption support.
EXPORT	Suitable for export	Because of export regulations, some CipherSuites are not suitable for export. Only CipherSuites that contain the word EXPORT are suitable for international use.
NULL	No encryption	Data is not encrypted.

Name	Defines	Description
DES 3DES DES40 RC4_40 RC4_128	Encryption algorithms	System: Key length: DES 56 3DES 168 DES40 40 RC4_40 40 RC4_128 128 The greater the key length, the greater the encryption strength.
EDE CBC	Encryption and decryption modes	CBC and EDE are modes by which DES algorithms are encrypted and decrypted.
SHA MD5	Hash function	SHA and MD5 are hash methods used to compute the message digest when generating a digital signature.

Note Browsers do not support anonymous cipher suites.

Security characteristics

There are four categories of security characteristics:

- **Simple** The predefined characteristics sybpks_simple and sybpks_simple_mutual_auth offer authentication but no encryption.
- **Strong** The predefined characteristics sybpks_strong and sybpks_strong_mutual_auth offer greater domestic encryption strength.
- **Domestic** All characteristics are suitable for domestic use. Clients using international cipher suites can connect to servers using domestic security characteristics.
- **International** Because of export regulations, only these characteristics are suitable for export:
  - sybpks_simple
  - sybpks_simple_mutual_auth
  - sybpks_intl
  - sybpks_intl_mutual_auth

Table 12-2 lists the name, the level of authentication, and the supported cipher suites for each security characteristic. Table 12-1 describes the cipher suites listed here.

Table 12-2: Security characteristics

Name of characteristic	Authenticates	Cipher suites
sybpks_simple	server	SSL_RSA_WITH_NULL_SHA SSL_RSA_WITH_NULL_MD5
sybpks_simple_mutual_auth	client/server	SSL_RSA_WITH_NULL_SHA SSL_RSA_WITH_NULL_MD5
sybpks_strong	server	SSL_RSA_WITH_3DES_EDE_CBC_SHA SSL_RSA_WITH_RC4_128_SHA SSL_RSA_WITH_RC4_128_MD5
sybpks_strong_mutual_auth	client/server	SSL_RSA_WITH_3DES_EDE_CBC_SHA SSL_RSA_WITH_RC4_128_SHA SSL_RSA_WITH_RC4_128_MD5
sybpks_intl	server	SSL_RSA_EXPORT_WITH_RC4_40_MD5 SSL_RSA_EXPORT_WITH_DES40_CBC_SHA SSL_RSA_WITH_NULL_SHA SSL_RSA_WITH_NULL_MD5
sybpks_intl_mutual_auth	client/server	SSL_RSA_EXPORT_WITH_RC4_40_MD5 SSL_RSA_EXPORT_WITH_DES40_CBC_SHA SSL_RSA_WITH_NULL_SHA SSL_RSA_WITH_NULL_MD5
sybpks_domestic	server	SSL_RSA_WITH_3DES_EDE_CBC_SHA SSL_RSA_WITH_RC4_128_SHA SSL_RSA_WITH_RC4_128_MD5 SSL_RSA_WITH_DES_CBC_SHA SSL_RSA_EXPORT_WITH_RC4_40_MD5 SSL_RSA_EXPORT_WITH_DES40_CBC_SHA SSL_RSA_WITH_NULL_SHA SSL_RSA_WITH_NULL_MD5
sybpks_domestic_mutual_auth	client/server	SSL_RSA_WITH_3DES_EDE_CBC_SHA SSL_RSA_WITH_RC4_128_SHA SSL_RSA_WITH_RC4_128_MD5 SSL_RSA_WITH_DES_CBC_SHA SSL_RSA_EXPORT_WITH_RC4_40_MD5 SSL_RSA_EXPORT_WITH_DES40_CBC_SHA SSL_RSA_WITH_NULL_SHA SSL_RSA_WITH_NULL_MD5
sybpks_domestic_anon	none	SSL_DH_anon_WITH_3DES_EDE_CBC_SHA SSL_DH_anon_WITH_RC4_128_MD5 SSL_DH_anon_WITH_DES_CBC_SHA The sybpks_domestic_anon profile is used for anonymous Diffie-Hellman communications. Neither the client nor the server is authenticated.

Defining security profiles

This section describes how to create, modify, and delete a security profile. All of the configuration tasks require you to first access the Security Profiles folder. To do this, highlight the Security Profiles folder from EAServer Manager.

See Table 12-3 on page 143 when creating, modifying, or deleting a security profile.

❖ **Creating a new security profile**

1 Select File | New Security Profile.

2 Enter the name of the new security profile. Click Create New Security Profile.

3 Complete the Security Profile sheet. Click Advanced to modify the default settings for the advanced SSL settings. Click Save. See Table 12-3 for a description of the security profile properties.

If you are using an Entrust ID, select the Use Entrust check box. Click the Entrust Tab and provide the Entrust information required to access your Entrust ID.

The new security profile now appears on the right side of the window when the Security Profiles folder on the left side of the window is highlighted.

❖ **Modifying an existing security profile**

1 Highlight the security profile you want to modify.

2 Select File | Properties.

3 Modify the properties. Click Save when finished. See Table 12-3 on page 143 for a description of the profile properties.

❖ **Deleting a security profile**

1 Highlight the profile entry you want to delete.

2 Select File | Delete Security Profile.

Table 12-3: General, advanced, and Entrust profile properties

Property	Description	Comments/example
Name	The name you give to the security profile.	
Description	A description of the security profile.	

Property	Description	Comments/example
Use Entrust	Select this check box to use an Entrust ID instead of a certificate contained in the Sybase PKCS #11 token.	Selecting this check box prevents access to the certificates contained in the Sybase token.
Security Characteristic	Select a name from the drop-down list of predefined security characteristics to use for this profile.	See Table 12-2 on page 142 for a description of security characteristics and the CipherSuites they support.
Description	A description of the selected security characteristic.	Each security characteristic comes with a description of its features.
Sybase PKCS #11 Token Certificate Label	From the drop-down list, enter the certificate label you want to use for this security profile. If you have not provided the PIN for the Sybase PKCS #11 token, you are prompted for one. This is the same PIN that you enter to access the EAServer Manager | Certificates folder.	If you are using an Entrust ID and click the Use Entrust check box, this property does not appear. See Chapter 13, "Managing Keys and Certificates" for more information on certificates.
SSL Cache Size	The number of entries in SSL session cache maintained by the server. The default cache size is 30.	See "SSL session caching and reuse" on page 145.
SSL Session Share	The number of concurrent connections that can simultaneously use the same session entry (ID) in the session cache. The default session share size is 10.	See "SSL session caching and reuse" on page 145.
SSL Session Linger	The duration for which a session entry is kept in the SSL session cache after the last SSL session using this session ID was closed. The default session linger value is eight hours.	See "SSL session caching and reuse" on page 145.
Log SSL Errors	When selected, additional information about SSL errors is logged.	
Set Defaults	Select the Set Defaults check box to restore all of the advanced settings to their default levels.	
Specify the Entrust INI File	Enter the complete path to the Entrust initialization file.	You can use the browse feature to locate this file. For example, on Windows, *%SystemRoot%\entrust.ini*.
Entrust User Profile	Enter the complete path to the Entrust user profile file.	You can also use the browse feature to locate this file. There is no default.
Entrust Password	The password to the Entrust login for this Entrust user profile.	

Property	Description	Comments/example
Allow non-Entrust client	Click this check box to allow non-Entrust clients to connect to listeners that use an Entrust ID.	

SSL session caching and reuse

For improved performance, EAServer caches SSL session identifiers and allows clients to reuse them. Since creating an SSL session requires CPU-intensive computations, SSL session reuse results in a relatively large performance gain over setting up completely new security sessions for each connection. The settings on the Advanced tab control how SSL clients can reuse sessions for subsequent and simultaneous connections.

Cached sessions allow the client to reuse a session in a subsequent connection. The SSL Cache Size setting controls how many entries can be cached. Set this to a number less than or equal to the maximum connections setting for the server. The cache requires approximately 64 bytes per entry. The SSL Session Linger value specifies how long cached session IDs remain valid.

The SSL Session Share setting specifies how many simultaneous connections can share one session ID. Session sharing can improve performance when the client opens multiple connections simultaneously. For example, a browser client may open several connections at once to download images linked to an HTML page. Session sharing allows the client to reuse the session for the second and subsequent connections, up to the number of concurrent connections specified by the SSL Session Share value.

Note These are advanced SSL parameters. They should be set only by someone who is knowledgeable about SSL.

Configuring listeners

A listener is an EAServer port that communicates to clients using various protocols. For protocols that use SSL security features (HTTPS and IIOPS), you assign a security profile to the listener. The profile defines security characteristics of the listener. For protocols that do not use SSL (HTTP, IIOP, and TDS), no security profile is required.

This section describes the tasks required to configure listeners. You can:

- Create a new listener and assign a profile to it.
- Assign a profile to an existing listener.
- Modify listener settings for both secure (IIOPS and HTTPS) and unsecure protocols (TDS, IIOP, and HTTP).

Preconfigured listeners

EAServer comes with preconfigured listeners for all protocols. Secure protocols are assigned a predefined security profile.

The default settings for the preconfigured listeners are described in Table 12-4. Only secure listeners use security profiles.

Table 12-4: Default listener settings

Listener name	Port	Security profile
http	8080	
https1	8081	sample1
https2	8082	sample2
iiop	9000	
iiops1	9001	sample1
iiops2	9002	sample2
tds	7878	
OpenServer	7979	

The default host for these listeners is “localhost.” Sybase recommends that after you install EAServer, log in to EAServer Manager and change the default host setting to the actual host name or IP address of your machine. If you do not, only connection requests originating from the EAServer host machine are accepted. This means that, until you modify your settings, EAServer Manager must also be on the same machine as the server. You can also modify port number settings for the preconfigured listeners. For more information, see “Configuring listeners” on page 145.

The OpenServer listener is intended for migrating existing Open Server applications to EAServer. See the *EAServer Programmer's Guide* for more information.

Note You must restart EAServer for your changes to take effect. If you have changed the server's host name and port number, you must also restart EAServer Manager and reconnect to the server using the new host name and port number.

Listener failover

If a server cannot retrieve listener information from the repository for an IIOP listener or if an IIOP listener has not been configured, the server attempts to open a listener at this address:

```
IIOP: localhost, 9000
```

Listener start-up can fail if a port is already in use. You can verify the listener addresses in use by viewing the initial log entries in the *srv.log* file. If the log messages indicate a listener configuration problem, use EAServer Manager to connect to the indicated IIOP address and reconfigure the server's listener properties.

Configuring listener properties

This section describes how to create, modify, and delete a listener. All of the configuration tasks require you to first access the Listeners folder from EAServer Manager:

1 Double-click the Servers folder.

2 Double-click the server for which you want to create, modify, or delete a listener.

3 Click the Listeners folder on the left side of the window.

❖ **Creating a new listener**

1 Select File | New Listener.

2 Enter the name of the new listener, then click Create New Listener.

3 Complete the information in the Listener Info window. See Table 12-5.

The new listener appears on the right side of the window when you highlight the Listeners folder.

❖ **Modifying an existing listener**

1 Highlight the listener you want to modify.

2 Select File | Properties.

3 Make your modifications and click Save. Listener properties are described in Table 12-5.

❖ **Deleting a listener**

1 Highlight the listener you want to delete.

2 Select File | Delete Listener Profile.

Table 12-5: Listener profile properties

Property	Description	Comments/example
Protocol	Select the protocol from the drop-down list: • HTTP • IIOP • TDS • HTTPS • IIOPS	HTTPS and IIOPS are secure protocols that provide all of the security features made available by SSL, including authentication and encryption. TDS, IIOP, and HTTP do not provide encryption. TDS and IIOP provide user name and password-based authentication.
Host	The name or IP address of the EAServer host to which the listener is being assigned.	For predefined listeners, change the initial setting from "localhost" to the actual machine name or IP address. This allows clients from other machines access to EAServer. **Note** Sybase recommends that you provide the IP address of the host instead of the host name. In certain cases, a client may not be able to resolve a host name; for example, the client's DNS server or *hosts* file may not have an entry for the specified host.
Port	The port number on the host to which the listener is assigned.	Make sure that the port is not in use by any other service.
Jaguar Security Profile	Select one of the preconfigured security profiles from the drop-down list. This field is enabled for only the secure protocols (HTTPS or IIOPS).	You can create new security profiles that can be assigned to a listener. See "Configuring security profiles" on page 139 for information on security profiles.

Property	Description	Comments/example
Enable Open Server Events	When selected, the TDS port accepts open server client connections, if not, only MASP requests are accepted.	You must use TDS as the protocol for Open Server events.

Configuring identities

Identities define a user name, password, and SSL session characteristics to be used by components or servlets that call other components. Identities are also used for inter-server authentication when propagating caller credentials in a call sequence that involves multiple servers. EAServer provides a System and Anonymous identity by default.

❖ **Defining a new identity**

1 Highlight the Identities folder.

2 Select File | New Identity.

3 Enter a name for the identity.

4 Configure the identity as described in “Configuring identity properties” on page 149.

❖ **Modifying or deleting an identity**

1 Expand the Identities folder and highlight the icon for the identity of interest.

2 Choose File | Delete to delete the identity, or choose File | Identity Properties to display the Identity Properties window.

3 If modifying the identity, make your modifications as described in “Configuring identity properties” on page 149 and click OK.

Configuring identity properties

The Identity Properties dialog has these tabs:

- “Identity properties/basic” on page 150 defines the user name and password.

- "Identity properties/SSL" on page 150 specifies whether connections made using the identity will use SSL and if so, the SSL session characteristics.
- "Identity properties/Entrust" on page 150 configures Entrust specific properties for SSL connections.

Identity properties/basic

Enter the user name and password for inter-server connections made using the identity.

Identity properties/SSL

Settings on this tab specify whether connections made using the identity will use SSL and if so, the SSL session characteristics.

❖ **Configuring the SSL settings**

1 If SSL is not to be used at all, choose <none> for the security characteristic. Otherwise choose the characteristic that defines the required level of security. See Table 12-2 for descriptions of the security characteristics.

2 Check Use Entrust if your site uses Entrust for SSL certificate management and you wish connections made with this identity to use an Entrust certificate.

3 If the specified security characteristic requires mutual authentication, choose a client certificate.

Client certificate field may require a password If you have not connected to the EAServer Manager | Certificates folder, you are prompted for the Sybase token PIN when you put the focus on the Certificate Label field. You must connect to EAServer Manager | Certificates folder or enter the correct PIN before you can view certificate names.

Identity properties/Entrust

If you enabled Entrust support in the SSL tab, the Entrust tab settings specify the Entrust certificate to be used.

❖ **Configuring the Entrust settings**

1. Browse to or type the path to the *entrust.ini* file (typically located in the Windows installation directory on Windows machines, and in the Entrust *clients* subdirectory on UNIX systems.
2. Browse or type the path to the Entrust profile file (*.epf* extension) that defines the certificate to be used.
3. Enter the password required to use the specified Entrust profile.

CHAPTER 13

Managing Keys and Certificates

This chapter describes how to use EAServer Manager | Certificates folder to manage keys and certificates for SSL security in EAServer.

SSL overview

You can configure EAServer to accept client connections over secure protocols IIOPS and HTTPS using:

- EAServer Manager | Certificates folder to manage key pairs and certificates for EAServer.

  See “Managing keys and certificates on EAServer” on page 154 for more information about EAServer Manager | Certificates folder.

- EAServer Manager to define security profiles that establish various levels of security on EAServer and assign them to a listener. Profiles allow you to determine:
  - Client and server authentication requirements
  - Encryption and decryption algorithms

  See “Configuring security profiles” on page 139 and “Configuring listeners” on page 145 for information on establishing security profiles and assigning them to EAServer listeners.

- EAServer to use certificates and listeners to authenticate clients, if necessary, and encrypt and decrypt data.

Managing keys and certificates on EAServer

EAServer Manager | Certificates folder allows you to manage keys and certificates used by EAServer.

EAServer Manager | Certificates folder management

This section describes the tasks involved in accessing and managing the server certificate database or the certificate database used by client applications. To manage the server certificate database, configure the top-level Certificates folder in EAServer Manager, while connected to the server. To manage the client certificate database, you must run the standalone Security Manager. Other than the tool used, the management tasks are identical for the client and server certificate database.

You can install and use the standalone Security Manager on a client machine to manage client keys, certificates, and trust information in a local database. The standalone Security Manager is completely independent of EAServer Manager and server installations. Except for the login screen, the standalone Security Manager is identical to EAServer Manager | Certificates folder used to manage server keys and certificates.

The Standalone Security Manager allows C++ CORBA clients and Java applications to access servers using SSL features over IIOPS connections. For more information, see these chapters:

- Chapter 5, "Using SSL in Java Clients"
- Chapter 6, "Using SSL in C++ Clients"
- Chapter 8, "Using SSL in ActiveX Clients"

❖ **Accessing the server certificate database in EAServer Manager**

To begin managing the server certificate database:

1 Start EAServer Manager as described in "Using EAServer Manager" in the *EAServer System Administration Guide*.

2 Expand the top level Certificates folder. The first time you put the focus on this folder in your session, you must enter the PIN for the PKCS #11 token. The default for new installations is “sybase”.

❖ **Starting the standalone Security Manager**

1 Change to the EAServer *bin* subdirectory.

2 Run sasecmgr to start Sybase Central.

3 In Sybase Central, choose Tools | Connect.

4 Choose Security Manager.

5 Enter the PIN for the PKCS #11 token. The default for new installations is “sybase”. Make sure the Client Root setting matches the installation you want to configure; this field should match the value of the JAGUAR or JAGUAR_CLIENT_ROOT environment variable as set for the installation to be configured.

❖ **Changing the user PIN**

The initial PIN for the PKCS #11 token is “sybase”. You can also use the same PIN to log in to EAServer Manager | Certificates folder and, if installed, the Sybase PKCS #11 token in Netscape. To change to a more secure PIN:

1 Select the Private Keys folder.

2 Select File | Change PIN.

3 Enter and verify the new PIN.

Restart Netscape for the new PIN to propagate to the Sybase PKCS #11 token.

❖ **Displaying PKCS #11 module information**

1 Select the Private Keys folder.

2 To view information about the Sybase PKCS #11 module, including the library version and the Cryptoki version, select File | Module Information.

To view information about the Sybase PKCS #11 token that manages your key and certificate information, including status and version information, select File | Token Information.

❖ **Logging out of the PKCS #11 module**

1 Select the Private Keys folder.

2 Select File | Logout.

You are still logged in to EAServer Manager but can no longer access keys or certificates.

Test CA management

The test CA is a signing authority that signs user certificate requests. These certificates can be used by clients and EAServer to test the security features of your applications. Certificates signed by the test CA are not intended for commercial applications. If you already have an in-house CA or other signing authority, you may not need to use the test CA.

Note The test CA must exist before you can access the Process Certificate Request and Generate User Test Certificate options.

❖ **Creating a test CA**

To verify that the test CA is available, highlight the CA Certificates folder. You should see the Sybase Jaguar User Test CA on the right side of the window. If not, you must generate the test CA.

1 Select the CA Certificates folder.

2 Select File | Generate Test CA.

The Sybase Jaguar User Test CA displays on the right side of the window. You can now generate test certificates signed by the test CA and process certificate requests.

❖ **Generating a user certificate signed by the test CA**

1 Select the CA Certificates folder.

2 Select File | Generate User Test Certificate. The Generate User Test Certificate wizard displays.

3 Supply the required information described in Table 13-1. Click Back and Next to review and modify information.

4 You can use any of the following characters for the label:

- Letters A – Z and a – z
- Numeric values 0 – 9
- (space) ’ ( ) + , - . / : = ?

5 Click Finish to exit the wizard and generate the certificate.

6 Click OK in the Info dialog. The certificate displays when you highlight the User Certificates folder.

Table 13-1: User test certificate information

Property	Description	Comments/example
Key Strength	Select the authentication key strength. The greater the number, the stronger the encryption. Your options are: • 512 bits • 768 bits • 1024 bits	For international users, key strength is 512.
Key Label	The name that identifies the certificate.	Required field. The label must be unique among all labels used for all certificates.
Validity Period	From the drop-down list, select the length of time that the certificate is valid.	When a client (or server) presents a certificate for authentication, EAServer (or the browser) checks to see if the certificate has expired.
Cert Usage	Click the check box for either or both: • SSL Client • SSL Server	The same certificate can be used by a client and/or EAServer.
Common Name	Your first and last name.	Required field.
User ID	Any ID that would further identify you.	
Organization	The name of your company, university, or other organization.	Required field.
Organization Unit	The name of a department within your organization.	
Locality	The location of your organization.	You must supply at least one of: • Locality • State/Province • Country
State/Province	State or province name.	
Country	Your two-digit country code; for example, “U.S.”	
Requester Name	The person requesting the certificate.	
Server Admin	The name, if any, of the server administrator.	
E-Mail	Your e-mail address.	

Property	Description	Comments/example
Mark Private Key Exportable	Checked by default, this property allows you to export this certificate along with its private key.	See "Installing and exporting certificates" on page 165 for more information. **Note** If checked, you can later uncheck this property. Once unchecked, you cannot change this property. If unchecked, you cannot export this certificate and private key.

❖ **Processing a certificate request**

EAServer Manager | Certificates folder can process a certificate request generated from elsewhere. The test CA signs the request and generates the certificate.

1 Select the CA Certificates folder.

2 Select File | Process Certificate Request.

3 Paste the certificate request into the window as indicated. Here is an example of a base64 certificate request. You must include the entire contents, including the BEGIN and END lines:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIH4MIGjAgEAMD4xCjAIBgNVBAMTAWExCjAIBgNVBAoTAWExCjA
IBgNVBAcTAWEx
CzAJBgNVBAgTAmNhMQswCQYDVQQGEwJ1czBcMA0GCSqGSIb3DQE
BAQUAA0sAMEgC
QQC9Yn9AOzflqIarPCC7eRdr3C0wrIG+3B2T+pEs9sdgEjnc/bw
1GfxcZKYamWXg
G1KQycFqkdrFNP79fgRCOd3xAgMBAAGgADANBgkqhkiG9w0BAQQ
FAANBAIEljmCB
HbFdNj0MtFDa002f/Trl6FtGCh7Gs23pZlWIUzDlGFowiuJY6iM
Dzd/1bJz5yYB+
IvlM9Ath/zTF2eY=
-----END NEW CERTIFICATE REQUEST-----
```

4 Set the following certificate properties:

- **Format Type** Identifies the format type of the request, either "base64" or "binary."
- **Cert Usage** Depending on how you will use the certificate, select SSL Client, SSL Server, or both.

- **Validity Period** Select the length of time that the certificate is valid.

5 Click Next. The certificate is generated and displays in the dialog. Here is the signed base64 certificate:

```
-----BEGIN CERTIFICATE-----
MIICYTCCAcqgAwIBAgIBBzANBgkqhkiG9w0BAQQFADCBgjEzMDE
GA1UEAxMqU3li
YXNlIEphZ3VhciBVc2VyIFRlc3QgQ0EgKFRFU1QgVVNFIE9OTFk
pMSAwHgYDVQQK
ExdTeWJhc2UgSmFndWFyIFVzZXIgVGVzdDEpMCcGA1UEBxMgU3l
iYXNlIEphZ3Vh
ciBVc2VyIFRlc3QgTG9jYWxpdHkwHhcNOTgwNzAyMDIzOTEzWhc
NOTgwOTAyMDIz
OTEzWjBHMQ0wCwYDVQQDEwR0ZXN0MQ0wCwYDVQQKEwR0ZXN0MQ0
wCwYDVQQHEwR0
ZXN0MQswCQYDVQQIEwJjYTELMAkGA1UEBhMCdXMwXDANBgkqhki
G9w0BAQEFAANL
ADBIAkEAvzvqs9yjW/PDCt/Rotp9x9PHrULLeGOLlVSubo9poY1
f5OYwsrjfaOtT
bkhWDrakuwJJk8smDNSAl93tdP9r8wIDAQABo2UwYzAMBgNVHRM
EBTADAQEAMB0G
A1UdDgQWBBTAT0n9qsvdfqc9NzGPA5oLKsMzJjAhBgNVHSMEGjA
YoBYEFGLT8qZb
3LtGjw84nxna9YBHb7q6MBEGCWCGSAGG+EIBAQQEAwIAwDANBgk
qhkiG9w0BAQQF
AAOBgQB3OStVqhoWT66yXNsrznCg9t8yNClobnKGOJTqt+VbhV7
BUgBH+fVSjf7v
xJyV4twwlBvU08PsKYQGj4sJ1Ao3lsOXWrr6YZIHZZ6p9P8JXjY
016Vg9g5SDmEV
jgGbwy6ZOZYx27npp4X31WXY27KDZrV/FrwvF6/Pv6mZY7ijUw=
=
-----END CERTIFICATE-----
```

6 Select Save to File and enter the full path name to save the generated certificate as a file. You can also select Browse to specify the location for the file.

If you want to use this certificate for authentication, you must install the certificate on the same machine that generated the certificate request, since this is where the private key is stored.

Note Certificates signed by the test CA are intended for testing only. In a real-life situation, the CA would verify user information to establish identity.

❖ **Exporting the test CA certificate**

You can export certificates, including the test CA certificate. Exporting the test CA certificate allows you to load it into Netscape 4.0*x* browsers and mark it trusted. This prevents Netscape from displaying warnings about untrusted certificate authorities when you use listeners that use certificates signed by the test CA.

1 Select the CA Certificates folder.

2 Highlight the Sybase Jaguar User Test CA.

3 Select File | Export Certificate.

4 From the Export Certificate wizard, select the format type for the exported certificate. For the Test CA, select Binary Encode X509 Certificate. Click Next.

5 Select Save to File and enter the full path name to a file that will contain the test CA.

Do not add any extension to the file name. A *.crt* extension is automatically added to the exported certificate. Netscape 4.0*x* recognizes this extension as a X.509 certificate and handles it accordingly.

6 Click Finish to export the certificate to the file you specified.

For general information about the Export Certificate wizard and certificate types, see "Installing and exporting certificates" on page 165.

❖ **Loading the test CA's certificate into Netscape 4.0*x***

You must be logged in to the Netscape token.

1 Enter the full path of the file that contains the exported test CA's certificate in Netscape's URL/Netsite field.

2 Select Open and click OK.

3 Click Install Certificate. Netscape recognizes the *.crt* extension as belonging to a certificate authority and displays a series of dialogs asking if you want to accept the CA.

If Netscape does not recognize the *.crt* file extension, perform these steps and restart Netscape before trying to load the test CA:

a From Netscape, select Edit | Preferences.

b Under Category, click Applications.

c Under Description, scroll down and select "Internet Security Certificate." Click Edit.

d Verify that the Mime Type field contains:

```
application/x-x509-ca-cert
```

e Click OK.

Note If you are using UNIX, make sure the following line is in your *~/.mime.types* file before you start Netscape:

```
application/x-x509-ca-cert      crt cer ber der
```

This line ensures that Netscape recognizes the *.crt* file extension.

---

4 Follow the instructions in the dialogs to accept this certificate.

Netscape now allows you to connect to EAServer ports that require authentication, and accepts the certificates signed by the test CA without displaying warnings.

Key management

This section describes the tasks involved in key management.

To view the private keys installed in the security module, select the Private Keys folder. The private keys display on the right side of the window.

EAServer Manager | Certificates folder displays any private key that does not have a certificate associated with it, including private keys that have an outstanding certificate request. For example, you may generate a key pair and request a certificate from a CA at the same time. It may take several days to receive your certificate. In the meantime, the private key displays when you highlight the Private Keys folder.

Sybase recommends that you delete any private key that does not have an outstanding certificate request associated with it.

❖ **Viewing information about a private-key**

1 Select the Private Keys folder.

2 Highlight the key whose information you want to view.

3 Select File | Key Information. The Key Information dialog box displays the length of the key.

❖ **Deleting a private key**

1 Select the Private Keys folder. The private keys display on the right side of the window.

2 Select the key that you want to delete.

3 Select File | Delete Key.

Certificate management

EAServer Manager | Certificates folder comes with several preinstalled CA certificates. EAServer accepts client certificates only if they have been signed by a trusted CA. You can modify the trust attribute for any of the preinstalled certificates. See “Viewing certificate, trust, and export information” on page 168 for more information.

❖ **Generating a key pair and requesting a certificate**

You can generate a key pair and send the certificate request to a CA to be signed. Once the CA has signed and returned the request, you can install the certificate.

1 Select the Private Keys folder.

2 Select File | Key/Cert Wizard.

3 Supply the required information, described in Table 13-2. Use Back and Next to review or change any information.

You can use any of the following characters:

- Letters A – Z and a – z
- Numeric values 0 – 9
- (space) ’ ( ) + , - . / : = ?

In Asian-language editions of EAServer, you can enter an Asian-language date in the Certificate Signing Request wizard in Security Manager. Before generating requests that contain UTF-8 characters, check with your certificate authority (CA) whether UTF-8 data is supported.

4 Click Finish to exit the wizard. EAServer Manager | Certificates folder generates the key pair and saves the certificate request to a file that you specify, or installs a certificate if you have pasted one into the certificate dialog.

5 Send your certificate request to a CA for signing. Depending on the CA, this could be through e-mail or by attaching to the CA's URL.

6 When you receive it, install the certificate. See "Installing and exporting certificates" on page 165.

The new private key appears on the right side of the window when you highlight the Private Keys folder. Once the certificate is received and installed, the private key is removed from the private key list.

Table 13-2: Certificate request information

Property	Description	Comments/example
Key Strength	Select the authentication key strength. The greater the number, the stronger the encryption. Your options are: • 512 bits • 768 bits • 1024 bits	For international users, key strength is 512.
Key Label	The name that identifies the private key/certificate.	Required field. The label must be unique among all labels used for certificates.
Mark Private Key Exportable	Check this box to allow the export of this certificate along with its private key.	See "Installing and exporting certificates" on page 165 for more information. **Note** If checked, you can later uncheck this property. Once unchecked, you cannot change this property. If unchecked, you cannot export this certificate and private key.
UTF-8 Encoding	Check this box to allow entry of UTF-8 encoded characters.	Allows entry of Asian-language text. Before generating requests that contain UTF-8 characters, check with your certificate authority (CA) whether UTF-8 data is supported.
Common Name	This could be your first and last name or name of a university or EAServer host name.	Required field.
User ID	Any user ID that would further identify you.	
Organization	The name of your company, university, or other organization.	Required field.
Organization Unit	The name of a department within your organization.	

Property	Description	Comments/example
Locality	The location of your organization.	You must supply at least one of: • Locality • State/Province • Country
State/Province	The name of your state or province.	
Country	Your two-digit country code; for example, "U.S."	
Requester Name	The person requesting the certificate.	
Server Admin	The name, if any, of the server administrator.	
E-Mail	Your e-mail address.	
Server Certificate Request	Displays the request information along with the generated public key.	Depending on the CA, you might be able to copy and paste the certificate request from this window into an e-mail and forward it for signing.
Save to File	Select this option and enter the full path name to save the generated certificate request as a text file. You can also use the browse feature to locate and save the file.	If you do not immediately send the certificate request to be signed, save the certificate request to a file and send it for signature later.
Cut and Paste the Certificate	If available, paste the signed certificate in this window for installation.	If you do not install the signed certificate now, you can use the Install Certificate option when you receive your signed certificate.
Format Type	Identifies the format of the certificate request. Your options are "base64" or "binary."	For server certificates, you would normally use a base64 format.

Certificate file extensions and types

When installing or exporting a certificate, EAServer Manager | Certificates folder determines the type of certificate based on the file extension. The extensions and the type of certificates they represent are:

- **.p7c** Belongs to a PKCS #7 certificate chain.
- **.crt** Belongs to X.509 certificates, including CA certificates. In addition, Netscape certificate chains end with a *.crt* extension.
- **.p12 and .pfx** Belong to transferred user certificates. Sybase's PKCS #12 implementation generates PKCS #12 files with a *.p12* file extension. This extension is recognized by both Netscape and Internet Explorer. The earlier PKCS #12 standard specified a *.pfx* file extension. You can install a PKCS #12 file that uses either extension into Sybase's PKCS #11 token.

- **Binary and base64** Certificates can either be encoded/decoded using a binary or base64 scheme. Base64 is based on an ASCII format and certificates of this type can be installed from a file or pasted into the appropriate window. Binary certificates, on the other hand, must be read from a file. The encoding scheme has no effect on a certificate's file extension.

Transferring versus importing and exporting: Transferring user certificates and private keys allows you to use the certificate and private key in the target security environment. Exporting, installing, and marking a CA certificate trusted in the target security environment simply allows you to accept certificates that have been signed by that CA.

❖ **Installing and exporting certificates**

EAServer Manager | Certificates folder allows you to export or import (install):

1 Certificates signed by the test CA.

2 Certificates signed by another CA.

3 Certificate chains – a certificate chain is a certificate that has been signed by a CA, which in turn has been signed by a CA, and so on. The certificate contains information that traces the path of the certificate back to the root CA (the original signer).

4 A signer's (CA) certificate. You need to install a signer's certificate and mark it as trusted so that EAServer accepts certificates signed by that CA.

5 User certificates and their corresponding private key using the PKCS #12 standard.

PKCS #12 is an RSA standard that specifies a transfer syntax for personal identity information. EAServer's support of the PKCS #12 standard allows you to move user certificates and private keys between systems and programs that support the PKCS #12 standard, such as Netscape Communicator and Microsoft's Internet Explorer.

Sybase's PKCS #12 implementation allows you to transfer certificates and private keys in either a domestic format (128-bit encryption) or international format (40-bit encryption). You can find more information about domestic and international support in "Configuring security profiles" on page 139.

❖ **Installing a certificate**

1 Select the folder that corresponds to the type of certificate you are installing.

2 Select File | Install Certificate.

3 Either paste the entire contents of the certificate into the box (base64 encoded certificates only), or click the Import from File box.

If you select Import from File, the cut and paste area is dimmed. Use the browse feature to locate the certificate.

4 Click Install. If the certificate is of type *.crt* or *.p7c*, it is installed. If the file is a PKCS #12 type (has either a *.p12* or *.pfx* extension) the PKCS #12 Certificate/Private Key window displays:

a Enter the password that allows access to the file. This is the password you entered when you exported the certificate and private key.

b To export the certificate and its private key at a later time you must check the Mark private key as exportable check box, which is, by default, already selected.

c Click Done.

The certificate is assigned to a folder based on its type:

- **User** Your certificates and other user certificates, including certificates signed by the test CA used to authenticate EAServer. These are the certificates that have a matching private key stored in the PKCS #11 token.
- **CA** Certificates obtained from CAs. These identify the signers of certificates that EAServer recognizes.
- **Trusted** A subset of the CA certificates. These are the signers of certificates that EAServer trusts. EAServer accepts the certificates from clients that have been signed by trusted CAs. You must mark a CA as trusted before it appears in the Trusted folder. See “Viewing certificate, trust, and export information” on page 168 for more information.
- **Other** Certificates obtained from other users or organizations that cannot be identified as User or CA.

Once installed, you can assign a user certificate to a security profile. For more information, see “Configuring security profiles” on page 139.

After installing a signer's certificate, mark it as trusted if you want to accept certificates signed by that signer. See "Viewing certificate, trust, and export information" on page 168 for more information.

❖ **Exporting a certificate**

1 Select the Certificates folder that contains the certificate to be exported.

2 Highlight the certificate to be exported.

3 Select File | Export Certificate.

4 From the Export Certificate wizard, select the format type of the certificate to be exported.

If you have chosen Export Certificate from the User Certificate folder, and you selected "Mark Private Key Exportable" when you generated the key pair and requested a certificate, the PKCS #12 option is available.

5 Depending on the type of certificate you select, one of two windows appears:

- If you have selected a certificate format that is *not* PKCS #12, select Save to File and enter the full path name to a file that contains the certificate.

  Do not add any extension to the file name. The appropriate extension is automatically added to the exported certificate.

- If you have selected PKCS #12, enter and confirm a password used to protect access to the exported certificate and its private key. When you try to install the certificate, you are prompted for this password; there are also several advanced options you can configure that affect the exported certificate. See "Advanced PKCS #12 options" on page 167. When you are finished, click Next.

  Select Save to File and enter the full path name to a file to contain the certificate.

  Do not add any extension to the file name. The appropriate extension is automatically added to the exported certificate.

6 Click Finish to export the certificate to the file you specified.

Advanced PKCS #12 options

The advanced screen allows you to modify the PKCS #12 options listed below. The default settings are appropriate in most cases and should only be modified by experienced users:

- **Include certificate trust chain** If the certificate is part of a chain, clicking this box adds information about the CAs in the certificate's chain. See "Verifying a certificate" on page 169 for additional information about certificate chains.
- **Private key encoding algorithm** The password-based algorithm used to protect the contents of the exported private key. The default algorithm is 40BitRC2, which is accepted by most browsers. If you want to export the private key using stronger or weaker encryption, select an algorithm from the drop-down list, but be sure that the target browser accepts the stronger encryption. EAServer Manager | Certificates folder can export or import private keys that are shrouded with any of the listed algorithms.
- **Certificate encoding algorithm** The password-based algorithm used to protect the contents of the exported user certificate. The default algorithm is 40BitRC2, which is accepted by most browsers. If you want to export the certificate using stronger or weaker encryption, select an algorithm from the drop-down list, but be sure that the target browser accepts the stronger encryption. EAServer Manager | Certificates folder can export or import user certificates that are shrouded with any of the listed algorithms. See "Configuring security profiles" on page 139 for a description of the various encryption methods and terms.

❖ **Viewing certificate, trust, and export information**

You can view the information about the certificates that you have installed and your own certificates, including identifying, trust, and usage information. To view certificate information:

1 Select the folder for the type of certificate you want to view:
   - User
   - CA
   - Trusted
   - Other
2 Select the certificate you want to view.
3 Select File | Certificate Info.

The Certificate Information dialog appears. Use the scroll bar to view all of the information.

The Certificate dialog includes a Trusted Certificate check box. Based on the policies of your organization, trustworthiness of the certificate signer, and other considerations, specify whether or not to mark a certificate as trusted. Only CA certificates can be marked as trusted or untrusted.

Certificates that are marked as trusted display when you select the Trusted folder.

For user certificates, an Exportable Private Key check box is provided. If this box is checked, you can export the certificate, along with its private key. To prevent future exports, you can uncheck the box. Once unchecked, the private key can never be exported. See “Installing and exporting certificates” on page 165 for more information.

❖ **Verifying a certificate**

EAServer Manager | Certificates folder verifies the signature, expiration date, and validity of a certificate. If the certificate is part of a chain of certificates, it verifies each certificate in the chain.

A chain involves more than one certificate. Each certificate in the chain is signed by the preceding certificate. For the certificate to be verified, the entire chain must be verified. If a peer offers a certificate for authentication that belongs to a chain, at least one CA within the chain must be trusted for the certificate to be accepted.

To verify a certificate:

1 Select the folder for the type of certificate you want to verify.

2 Highlight the certificate you want to verify.

3 Select File | Verify.

A dialog appears that either verifies the certificate or informs you that verification was unsuccessful. Do not use certificates that fail verification.

❖ **Renaming a certificate**

Only the label of the certificate is changed. The content of the certificate remains the same.

1 Select the folder type for the certificate you want to rename.

2 Highlight the certificate to rename.

3 Select File | Rename Certificate.

4 Enter the new name of the certificate. Click Done.

❖ **Deleting a certificate and its associated private key**

EAServer Manager | Certificates folder allows you to delete your own certificates and associated private keys, the test CA, and certificates that you have obtained from others.

1 Select the folder for the type of certificate you want to delete.

2 Highlight the certificate you want to delete.

3 Select File | Delete Certificate.

Note If you delete the test CA, certificates that were signed by the test CA are no longer useful. In this case, you need to generate a new test CA and new certificates signed by the new test CA to test your security scenarios.

Using Netscape to manage certificates on the client

PKCS #11 is an RSA standard that specifies an API called Cryptoki, which performs cryptographic functions, such as key-pair and certificate management.

Netscape 4.0*x* supplies a PKCS #11 module that allows you to manage the client-side certificates. Sybase also provides a PKCS #11 module that allows you to manage your certificates. Sybase recommends that you install the Sybase PKCS #11 module into Netscape, which provides immediate access to the EAServer sample server certificates.

Installing Sybase PKCS #11 into Netscape 4.0*x*

Start Netscape 4.0*x*, then:

1 Select Communicator | Security Info from the window. Or, you can click the Security icon (the padlock) in the tool bar.

2 Click on Cryptographic Modules.

3 Click Add. You see a new dialog, Create a New Security Module.

4 For Security Module Name, enter “Sybase PKCS”.

5 For Security Module File, type the full path to your *libjsybcki* file, then click OK. For example, on Windows, enter:

i:\Program Files\Sybase\Jaguar CTS\dll\libjsybcki.dll

On UNIX, enter:

/work/JagPKS/lib/libjsybcki_r.so

6 You should see a prompt asking for a Sybase password or PIN. Enter "sybase". If you do not see this prompt, verify the path to the DLL/shared object.

7 After entering the password, you see Sybase PKCS listed as a security module. Click on the Sybase PKCS module, then select View/Edit. A new window, the Edit Security Module window, displays. This window contains controls for the Sybase PKCS module.

8 Click "More Info" in the new window, and verify that the state is "Ready" in the Token/Slot Information window. Click OK to close the Token/Slot Information window.

9 You can change the Sybase PKCS module password by clicking Change Password in the Edit Security Module window. Click OK to close this window.

When both EAServer and Netscape run on the same machine, they share Sybase PKCS #11 database files. If you change the PIN, you must use the new PIN when you log in to either EAServer or Netscape. Sybase suggests that you change your PIN through EAServer Manager | Certificates folder, which automatically propagates the PIN changes to the security profiles. If you change the PIN through Netscape, you must also change the PIN in all of the security profiles; otherwise EAServer secure listeners using those security profiles may not start the next time you restart the server. See "Changing the user PIN" on page 155 for information about changing the PIN in EAServer Manager | Certificates folder.

If you modify the PIN through EAServer Manager | Certificates folder, you need to restart Netscape for the changes to take effect. If you modify the PIN through Netscape, while the server is running, shut down and restart the server. Supply the new PIN to connect to EAServer Manager | Certificates folder.

Obtaining a key pair and certificate

Sybase PKCS #11 includes two sample server certificates. You can use these certificates when communicating with EAServer from your browser. You can also request new certificates from your CA or certificate server and install them in your browser.

In general, the steps involved in obtaining a certificate from a CA are:

1 Initiate a certificate request operation by connecting to a Web server hosting the public-key infrastructure (PKI) administration HTML pages for the CA/PKI.

2 On the PKI administration Web page, complete the identity information form required by the CA.

3 Netscape generates a key pair, and stores the private key in the PKCS #11 module. The public key is digitally signed and forwarded to the CA.

   You must be logged in to the PKCS #11 module to store or access the private key.

4 The CA approves the request, generates the certificate, and makes the certificate downloadable by way of a URL.

5 The CA notifies you of certificate approval and URL location through e-mail.

6 Open the URL to get the certificate. You must log in to the PKCS #11 module using the previously established PIN. The browser automatically installs the certificate, locates the previously stored matching private key using the key ID, and sets the user-specified label.

SSL certificate information in servlets

This section describes how to include the client's certificate information into Java servlets that are hosted by EAServer.

You can obtain SSL certificate information about the client as follows:

```
java.security.cert.X509Certificate peerX509;
ServletRequest request;

peerX509 = (java.security.cert.X509Certificate)
       request.getAttribute
       ("javax.servlet.request.X509Certificate");
```

Where `request` is the `ServletRequest` parameter passed in the `doXXX()` method.

The ServletRequest technique is portable to other J2EE based application servers. See section 5.7 of the Java Servlet Specification version 2.2 for more information on these APIs.

You can also obtain the client's SSL certificate information using the EAServer CtsSecurity APIs, as follows:

```
import CtsSecurity.*;

CtsSecurity.X509Certificate peerX509;

peerX509 =
(CtsSecurity.X509Certificate)request.getAttribute(
"com.sybase.jaguar.servlet.request.X509Certificate");
```

Note Methods in `CtsSecurity.X509Certificate` and `java.security.cert.X509Certificate` are very similar. `java.security.cert.X509Certificate` documentation is available as part of the JDK documentation. `CtsSecurity.X509Certificate` documentation is available in the EAServer repository documentation.

CHAPTER 14

Entrust PKI Integration

Overview

EAServer integrates an Entrust public-key infrastructure (PKI) that enables servers and clients to use Entrust IDs for client/server authentication. To assign an Entrust ID (Entrust profile) to an EAServer listener:

1 Install and use Entrust/Entelligence software to manage Entrust keys and obtain an Entrust ID. See the Entrust documentation for more information.

2 Use EAServer Manager to configure a security profile that specifies the Entrust ID you obtained in step 1. You can configure the security profile to accept either non-Entrust clients or only clients that supply an Entrust ID. See “Defining security profiles” on page 143 for more information.

3 Assign the security profile to a listener. See “Configuring listener properties” on page 147 for more information.

In client applications, set the appropriate ORB properties to use Entrust IDs. This chapter describes server-side Entrust configuration. For client-side use of Entrust and non-Entrust certificates, see the following chapters:

- Chapter 5, “Using SSL in Java Clients.”
- Chapter 6, “Using SSL in C++ Clients.”
- Chapter 8, “Using SSL in ActiveX Clients.”

The current version of EAServer does not use Entrust encryption operations other than SSL signing by the private key.

For more information about Entrust, see their Web site at http://www.entrust.com.

Scenarios

There are three usage scenarios involving Entrust IDs and non-Entrust certificates:

- Both client and EAServer use non-Entrust certificates
- Entrust client and non-Entrust server (and vice versa)
- Both client and server use Entrust certificates

Both client and EAServer use non-Entrust certificates

In this scenario, you use EAServer's EAServer Manager | Certificates folder to access the Sybase PKCS #11 token to manage EAServer's keys and certificates. On the client, you use either the browser's mechanism to manage keys and certificates for Java applets or the standalone Security Manager to access the Sybase PKCS #11 token to manage keys and certificates for C++ and Java applications.

See Chapter 13, "Managing Keys and Certificates" for information about EAServer Manager | Certificates folder, the standalone Security Manager, and Netscape certificate management.

Entrust client and non-Entrust server (and vice versa)

In a mixed environment of Entrust IDs and non-Entrust certificates, each side (client and server) must import the other's CA certificate so that it will be recognized and accepted as coming from a trusted CA. For example, import the Entrust CA certificate into the non-Entrust server's PKCS #11 token using EAServer Manager | Certificates folder (the Entrust CA certificate is imbedded in the user profile's *.key* file). Mark the CA certificate trusted.

See Chapter 13, “Managing Keys and Certificates” for information about importing CAs and marking certificates as trusted.

You can then use the certificates and Entrust IDs as follows:

- **Client side** client applications establish security through the ORB/global property or callback feature.
- **Server side** to allow non-Entrust clients, select the allow non-Entrust client check box when you configure a security profile. See “Configuring security profiles” on page 139 for more information.

Both client and server use Entrust certificates

When both the client and server use Entrust IDs, use Entrust to manage the IDs and use EAServer Manager to establish a security profile that uses those IDs.

See “Defining security profiles” on page 143 for information on configuring security profiles to use either Entrust IDs or non-Entrust certificates and enabling non-Entrust clients to connect to a listener using Entrust IDs.

CHAPTER 15

Tutorial: Using SSL

In this tutorial, you will run an applet that uses the SSL security features supported by EAServer. This tutorial will familiarize you with how to create and manage security certificates using EAServer Manager | Certificates folder, and how to use EAServer Manager to define listeners that use these certificates and other SSL features.

Overview of the security tutorial

You should be familiar with SSL concepts and terms before you run this tutorial. Refer to Chapter 1, “Security Concepts” for an overview of SSL concepts and Chapter 13, “Managing Keys and Certificates” to learn how to use EAServer Manager | Certificates folder, and the standalone Security Manager.

This tutorial has three phases:

1 Setting up your browser – export a personal certificate signed by Jaguar’s test Certificate Authority (CA) certificate and import it in to a browser. You will use this certificate to authenticate yourself when you connect to EAServer listeners that require client authentication.

2 Setting up EAServer – use EAServer Manager | Certificates folder and to:

- Generate a user certificate signed by the test CA. This certificate is used for EAServer authentication.

- Create a security profile that uses the generated certificate. The security profile defines various aspects of a secure connection, encryption strength, whether the client and/or server require authentication, and so on.
- Define a listener and assign to it a security profile. This establishes the security parameters of an EAServer port.

3 Running the SSL sample applet – connect to the HTML page that contains the applet from your browser on a secure HTTPS listener and run the sample applet.

Tutorial requirements

To run the tutorial, you need:

- The EAServer software. For installation instructions, see the *EAServer Installation Guide*.
- A Netscape Navigator or Microsoft Internet Explorer Web browser that supports security certificates.

Note Other browsers that support security certificates may work, but have not been tested with this tutorial.

Setting up your browser

In this tutorial, your browser connects to EAServer through a listener that requires client authentication. This requires you to install a personal certificate in the browser that authenticates your identity.

To install a personal certificate in your browser:

1 Start the server, EAServer Manager, and connect to the Certificates folder.

2 Export a personal (user) certificate signed by the Jaguar test CA.

3 Import the user certificate to your browser.

Start the server, EAServer Manager, and connect to the Certificates folder

If the server is not already running, follow the instructions under "Starting the server" in the *EAServer System Administration Guide* to start the server.

If you are not connected to EAServer Manager, follow the instructions in "Using EAServer Manager" in the *EAServer System Administration Guide* to connect to EAServer Manager. After connecting, browse to the Certificates folder, double-click on it, then enter your certificate database PIN.

Obtain and install a personal certificate

You need a personal certificate installed in your browser before the sample applets can attach to EAServer listener ports that require client authentication.

There are a variety of ways to get a personal certificate:

- **Attach to an in-house CA** Supply the required information to request a personal certificate.
- **Use a public CA** You can obtain your certificate from any public CA. A number of public CAs are available through your browser. To request a certificate through a Netscape browser:

  a Click the Security icon on the tool bar.

  b Click Yours on the left side of the window. This displays a list of your certificates.

  c If no certificates are displayed, you need to get one. Click Get a Certificate. You see a Web page of public CAs.

  You need to obtain a certificate from a CA that EAServer recognizes, or use EAServer Manager | Certificates folder to install the CA's certificate and mark it trusted. In EAServer Manager | Certificates folder, click the Trusted CAs folder to display a list of the trusted certificate signers that EAServer recognizes.

  d Select a CA and follow the instructions to obtain your certificate.
- **Use the sample certificates** EAServer comes with two sample personal (user) certificates signed by the test CA that you can use to authenticate yourself when connecting to EAServer listeners that require client authentication.

For this tutorial, export a user certificate using EAServer Manager | Certificates folder and import it in to your browser.

❖ **Exporting the sample user certificate from EAServer**

1 In EAServer Manager | Certificates folder, highlight the User Certificates folder.

2 Highlight one of the sample certificates.

3 Select File | Export Certificate.

4 In the Export Certificate wizard, select the PKCS#12 formatted data option. This option exports the private key and the certificate so that you can import it in to a browser and use it to authenticate yourself. Click Next.

5 Enter and confirm a password. You need to provide this password when you import the certificate in to a browser. Click Next.

6 Click the Browse button on the wizard and enter the path and file name of the exported certificate. Do not supply an extension; *.p12* extension is automatically appended to the certificate. Click Finish.

An information box appears confirming that the user certificate has been successfully exported. Click OK.

❖ **Importing the sample user certificate in to Netscape**

1 In Netscape, click the security icon.

2 Highlight "Yours" to view your certificate.

3 Click the Import a Certificate button.

4 Locate and highlight the certificate you exported from EAServer Manager | Certificates folder. Click Open.

5 Enter the password you used when you exported the certificate.

6 The certificate is imported to Netscape. You can view and verify its validity.

When your browser connects to EAServer listeners that require client authentication, you can select this certificate when Netscape prompts you for a user certificate.

❖ **Importing the sample user certificate in to Internet Explorer**

1 In Internet Explorer, select View | Internet Options (version 4.0) or Tools | Internet Options (version 5.0).

2 Select the Content tab.

3 Click the Personal Certificates button (version 4.0) or the Certificates button (version 5.0).

4 Click the Import button. Enter the complete path and file name and password of the exported certificate (version 4.0) or follow the wizard instructions to locate the certificate and enter the password (version 5.0).

5 The certificate is imported in to Internet Explorer. You can view and verify its validity.

When your browser connects to EAServer listeners that require client authentication, you can select this certificate when Internet Explorer prompts you for a user certificate.

Setting up EAServer

In this section, you will create a user certificate that is signed by the test CA and used for server authentication. You will assign this certificate to a security profile, and assign the security profile to a listener.

❖ **Creating a user certificate from EAServer Manager | Certificates folder**

1 Highlight the CA Certificates folder.

2 Select File | Generate User Test Certificate.

3 Provide the information in the Generate User Test Certificate wizard as follows:

- **Key Strength** Select 512 from the drop-down list.
- **Validity Period** Select two months from the drop-down list. The validity period determines how long the certificate is valid. When EAServer authenticates itself using this certificate, Netscape examines the validity period to see if it has expired.
- **Key Label** Enter `Tutorial_cert` for the name that identifies the certificate.
- **SSL Server** Select this box since you will use this certificate to authenticate EAServer.
- **SSL Client** The same certificate can also be used by clients for authentication. Since this certificate will not be used to authenticate the client, do not select this box.

- **Mark Private Key as Exportable** Since you are not using this certificate on other systems, do not check this box.

4 Click Next. Provide your personal and site information as requested in the Certificate Request Information window. Refer to "User test certificate information" in Chapter 13, "Managing Keys and Certificates" for information on these fields.

5 Click Finish. EAServer Manager | Certificates folder generates a user certificate that is signed by the test CA. To view the certificate, highlight the Users Certificates folder.

Creating and assigning a security profile to a listener

In this section, you will define a new security profile, which includes a security characteristic. The security characteristic determines characteristics of the client-EAServer connection, such as:

- **Authentication** The security profile you create for this tutorial requires certificates for authentication from both the client and server.
- **Encryption** The strength and method of encryption. The security profile you create for this tutorial will not encrypt data.

❖ **Creating a security profile**

1 Double-click the EAServer Manager icon.

2 Click the Security Profiles folder.

3 Select File | New Security Profile.

4 Enter `user_test` as the name of the security profile and click Create New Security Profile.

5 Enter the information in the SSL tab of the Security Profile Properties window as follows:

- **Description** Enter `sample security profile` as the description of this security profile.
- **Use Entrust** Uncheck this box. You would check this box if you were using an Entrust ID for authentication.
- **Security Characteristic** Select sybpks_intl_mutual_auth from the drop-down list. A description of this security characteristic displays in the Description window.

Refer to “Configuring security profiles”in Chapter 12, “Security Configuration Tasks” for more information about security characteristics.

- **Certificate Label** Select Tutorial_cert from the drop-down list. This is the label of the certificate you created earlier. The security profile uses this certificate to authenticate EAServer. If you have not logged in to EAServer Manager | Certificates folder, you are prompted for a PIN.
- **PIN** Enter the password (PIN) and press ENTER. This is the same PIN that allows access to EAServer Manager | Certificates folder. The default PIN is `sybase`. If you have changed this PIN, enter the new PIN. See Chapter 12, “Security Configuration Tasks” and Chapter 13, “Managing Keys and Certificates” for more information.

6 Click Save. EAServer Manager displays the new security profile.

You can now assign the user_test security profile to a listener.

See “Configuring security profiles” on page 139 for more information.

Assign a security profile to a listener

A listener identifies EAServer ports that accepts connection requests from clients using the following protocols:

HTTP
HTTPS
IIOP
IIOPS
TDS

When you define a listener, you choose a port number, the protocol, and, for secure protocols IIOPS and HTTPS, assign a security profile.

❖ **Assigning the test_profile security profile to a listener**

1 Double-click the EAServer Manager icon.

2 Double-click the Servers folder.

3 Double-click the Jaguar icon.

4 Click the Listeners folder.

5 Select File | New Listener.

6 Enter `https3` for the listener name and click Create New Listener.

7 When you see the Listener info window, supply the following:

- **Protocol** Select HTTPS from the drop-down list. You will use HTTPS as the protocol to retrieve the HTML page that contains the sample applet.
- **Host** Enter the name of the EAServer host.
- **Port** Enter the port number on the host machine for this listener. If not in use by any other service, enter `8083`.
- **Jaguar Security Profile** Select the user_test security profile from the drop-down list.

8 Click Save.

9 Restart EAServer:

a Highlight the server to which this listener belongs.

b Select File | Restart.

You now have a listener that accepts HTTPS connection requests at port 8083 (https://*hostname*:8083) and requires client and server authentication.

See “Configuring listeners” on page 145 for more information.

Running the SSL sample applet

The SSL sample applet contains code for both a Java and a C++ server component. The applet instantiates and runs the JUserCredentialTest (Java) or CUserCredentialTest (C++) component. The component retrieves and the applet displays information about the client certificate.

Complete instructions for running the applet are in the file *html\classes\Sample\SecurityDemo\readme.html* in your EAServer installation directory.

❖ **Using *readme.html***

1 Import the SecurityDemo package into EAServer Manager.

The SecurityDemo package contains a Java and a C++ component. These components both implement the SecurityDemo::UserCredentialTest interface.

2 Generate stubs and skeletons.

You need to generate the stub files for the Java applet and the skeleton files for the server component.

3 Compile the Java source files.

4 Run the applet.

You are instructed to load the HTML page that contains the applet at port 8080. If you connect to port 8080, authentication requirements are determined by the IIOP listener to which the applet connects.

You can also load the HTML page by connecting to the HTTPS listener port 8083 (https://*hostname*:8083) that you created earlier. Before the browser loads the page, you need to accept EAServer's certificate and supply the user certificate that you imported in to your browser for client authentication.

The SSL sample applet connects to the preconfigured IIOP listeners, IIOP at port 9000, IIOPS at port 9001, and IIOPS at port 9002. For the applet to run successfully, verify that the host name for these listeners is the same as the host name for the HTTPS listener (8083). Refer to "Preconfigured listeners" in Chapter 12, "Security Configuration Tasks" for more information.

Debugging the SSL sample applet

If you have difficulty running the sample:

- View the *srv.log* file to verify that the listeners are running.
- Check the Java console in your browser for error messages. To view the console, select Communicator | Java Console.
- If the *srv.log* or Java console indicates an untrusted certificate error, make sure you have loaded the test CA's certificate from EAServer Manager | Certificates folder in the browser. If you use a personal certificate signed by a CA other than the Jaguar test CA, make sure you have installed the signer's certificate (of your personal certificate) in EAServer Manager | Certificates folder.
- Make sure that the listener's Host Name field for all preconfigured listeners and the listener you created for this tutorial are set to the actual name or IP address of the host and *not* `localhost`.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

T

U

V

W